The AORTA Reasoning Framework - Adding Organizational Reasoning to Agents

Jensen, Andreas Schmidt

Publication date:
2015

Document Version
Publisher's PDF, also known as Version of record

PhD Thesis

DTU Compute
Department of Applied Mathematics and Computer Science

The AORTA Reasoning Framework

Adding Organizational Reasoning to Agents

Andreas Schmidt Jensen

Kongens Lyngby 2015
PHD-2015-372

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Matematiktorvet
Building 303B
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

PHD-2015-372
ISSN: 0909-3192

To my wife, Eva and our daughters, Rebekka and Leonora;
without you, I would have finished much earlier.

aorta, *n*. the main artery supplying blood from the heart to the rest of the body.

Oxford English Dictionary, Tenth Edition, 2005.

Summary

Intelligent agents are entities defined by, among other things, autonomy. In systems of many agents, the agents' individual autonomy can lead to uncertainty since their behavior cannot always be predicted. Usually, this kind of uncertainty is accommodated by imposing an organization upon the system; an organization that defines expected behavior of the agents and attempts to restrict the agents' behavior to let it match the expectations. Restrictions can lead to a decrease in autonomy, contradicting one of the pillars of intelligent agents.

This thesis presents the AORTA reasoning framework, which is a practical component (founded in logic) that enriches intelligent agents with organizational reasoning capabilities. We take the agent's perspective by devising a component that integrates with the agent's usual reasoning capabilities in a non-intrusive way. This results in agents that are both organization-aware and autonomous. The reasoning component makes them organization-aware, and their autonomy is intact because the component does not change the existing reasoning mechanisms. As such, it allows the agents to decide whether to adhere to the system's expectations.

The ability to reason about organizations has previously been successfully integrated into agent programming languages. However, the operationalization of an organization is usually tailored to a specific language. This makes it hard to apply the same approach to other languages and platforms. The AORTA reasoning framework distinguishes itself by being a generic framework that allows different kinds of agents to reason about different kinds of organizations.

We present our results in three main parts. In the first part, we present the theoretical foundations for the AORTA framework, which consists of semantics of norms, an organizational metamodel, and the AORTA reasoning component. The reasoning component is characterized by being completely decoupled from the cognitive agent, by its automated reasoning about norms and organizational options, and by the reasoning rules specified by the designer to act upon norms and options. We specify the reasoning component using structural operational semantics providing us with a formal, rigid description of the behavior of the component during execution. This enables us to precisely specify each reasoning phases (using transition rules), and it makes the implementation of the system quite straightforward.

The second part moves from theory to practice: we present an implementation of the framework and integrate it into various agent platforms. We show that the same configuration of the component can be used for different agent platforms, providing evidence for its use a general tool for organization-awareness. Furthermore, we use practical verification to show various properties of an implementation of agents and of the system in general.

In the last part, we discuss a potential issue with our framework. The possibility to commit to organizational objectives can affect the agent's autonomy, which contradicts our main goal. We propose a model that solves this problem by adding a filter to the agent's decision procedure that takes consequences of fulfilling a goal into account before deciding to commit to it. By considering both the agent's preferences and the expected outcome of fulfilling the goal, we show that it was possible for the agents to make qualified context-dependent decisions.

We claim that by using the AORTA reasoning framework, agents become organization-aware. The reasoning component provides capabilities to reason about organizations and our decision procedure ensures that the autonomy of the agents is still intact.

Resumé

Intelligente agenter forstås ofte som enheder der kan handle uafhængigt af andre – autonomt. I et system bestående af mange autonome agenter kan den enkelte agents opførsel ikke altid forudsiges, hvilket kan lede til usikkerhed omkring systemets formåen. Normalt imødekommes denne usikkerhed ved at indføre en organisation, som fastlægger systemets forventning til agenterne og forsøger at indskrænke agenternes muligheder, således at de holder sig inden for rammerne af organisationen. Disse indskrænkninger kan føre til mindre autonomi for den enkelte agent, hvilket modsiger en af grundpillerne for intelligente agenter.

I denne afhandling præsenteres AORTA: en platform til udvikling af agenter der kan ræsonnere om organisationer. Platformen består af en komponent (med en logisk kerne) der giver agenter denne evne til at ræsonnere om organisationer. Ved at tage agentens perspektiv udvikler vi en komponent der kan integreres med agenten uden at ændre dens sædvanlige ræsonneringsmekanisme. På denne måde skabes en agent der er bevidst omkring organisationer ("organization-aware") og er autonom. Vores komponent gør agenterne bevidste om organisationen, og da komponenten ikke ændrer agenternes eksisterende ræsonneringsmekanisme, er deres autonomi intakt. Således lader vi agenterne beslutte hvorvidt de vil imødekomme systemets forventninger.

Evnen til at ræsonnere om organisationer er før blevet integreret med agent-programmeringssprog. I disse tilfælde har implementeringen dog været skræddersyet til det specifikke sprog, og det har derfor været svært at anvende samme metoder i andre sprog og på andre platforme. AORTA udmærker sig ved at være en generisk platform, der muliggør at agenter på forskellige platforme kan ræsonnere om forskellige typer organisationer.

Vi præsenterer vores resultater i tre dele. I første del præsenterer vi det teoretiske fundament for AORTA, hvilket består af semantikken for normer, en metamodel for organisationer og vores ræsonneringskomponent. Komponenten er karakteriseret ved at den er fuldstændig adskilt fra agenten, ved automatiseret ræsonnering om normer og organisationsmæssige muligheder, samt ved ræsonneringsregler som gør det muligt at handle på de genererede muligheder. Vi specificerer komponenten ved hjælp af strukturel operationel semantik, hvilket giver os en formel beskrivelse af komponentens opførsel under udførelsen. Dette gør det muligt at specificere de enkelte faser (ved hjælp af overgangsregler) og det simplificerer den senere implementering af systemet.

I anden del af afhandlingen bevæger vi os fra teori til praksis: vi præsenterer en implementering af platformen og viser at den kan integreres med forskellige eksisterende agentplatforme. Vi viser at den samme konfiguration af komponenten kan genbruges på forskellige platforme, hvilket betyder at systemet er velegnet som et generelt værktøj til at gøre agenter bevidste om organisationer. Ydermere anvender vi verifikation til at vise en række egenskaber ved en implementering af agenter og ved systemet generelt.

I den sidste del diskuterer vi et potentielt problem ved vores løsning. Vores platform gør det muligt for agenter at forpligte sig til at løse organisationens mål, hvilket kan have betydning for agenters autonomi. Dette kan betyde at platformens funktion modsiger vores primære mål. Vi foreslår en model der løser dette problem ved at tilføje filter i agentens beslutningsprocedure. Dette filter tager højde for konsekvenserne ved at udføre en opgave før opgaven løses. Ved at betragte en agents præferencer samt det forventede udfald ved at udføre opgaven viser vi at agenten kan tage kvalificerede kontekstafhængige beslutninger ved at benytte dette filter.

Vi hævder at agenter bliver bevidste omkring organisationer ved at benytte AORTA. Vores komponent forsyner agenterne med kapaciteten til at ræsonnere om organisationer, og vores beslutningsprocedurer sikrer at deres autonomi forbliver intakt.

Preface

When my MSc supervisor, Associate Professor Jørgen Villadsen, contacted me about applying for a PhD scholarship, it did not require much thought. I was working in the private sector as a software engineer, but I missed having the possibility to immerse myself in specialized topics as I did during my MSc study. This was a perfect opportunity to study multi-agent systems and (hopefully) put a dent in the universe.

I decided to focus on *Organization-Oriented Programming in Multi-Agent Systems* (also the working title for the study), which was a natural continuation of my MSc thesis on organization-oriented and agent-oriented methods for programming agents. During my study I have touched upon decision making, belief revision, agent-oriented and organization-oriented programming languages, organizational modeling, meta logic, and many other concepts. I even had the pleasure of participating twice in the annual multi-agent programming contest (we came in second place both times).

One of my personal goals during my PhD was to make a *practical* contribution to the field. While I appreciate theoretical results, I really enjoy putting those results into practice. For example, to show that my framework theoretically makes agents organization-aware is not as fulfilling (for me) as actually implementing the framework and watching the agents work. The main result of my thesis (which is also the title of this work) – the AORTA reasoning framework – combines these points perfectly. AORTA (which stands for **A**dding **O**rganizational **R**easoning **T**o **A**gents) represents both the theoretical foundations and the implementation (and integration) of the system.

This thesis documents the results of my PhD study carried out in the section for Algorithms, Logic and Graphs in the Department of Applied Mathematics and Computer Science, Technical University of Denmark. The results presented were obtained in the period from March 2012 to May 2015. My supervisor was Associate Professor Jørgen Villadsen. In April and May 2013, I was on an external research stay at Delft University of Technology under the supervision of Virginia Dignum. I was on parental leave from January to March 2013 and again in August 2014.

The thesis is a monograph, which consists of material from the following previously published articles, extended and rewritten to form a coherent story:

- Jensen, A. S. (2013a). “Deciding between conflicting influences”. In: *Engineering Multi-Agent Systems*. Ed. by Cossentino, M. et al. Vol. 8245. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 137–155.
- Jensen, A. S., Aldewereld, H., and Dignum, V. (2013a). “Dimensions of organizational coordination”. In: *Proceedings of the 25th Benelux Conference on Artificial Intelligence*, pp. 80–87.

- Jensen, A. S. (2013b). “On programming organization-aware agents”. In: *Twelfth Scandinavian Conference on Artificial Intelligence, SCAI 2013, Aalborg, Denmark, November 20-22, 2013*. Ed. by Jaeger, M. et al. Vol. 257. IOS Press, pp. 287–290.
- Jensen, A. S. and Dignum, V. (2014). “AORTA: adding organizational reasoning to agents”. In: *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS ’14, Paris, France, May 5-9, 2014*. Ed. by Bazzan, A. L. et al. IFAAMAS/ACM, pp. 1493–1494.
- Jensen, A. S., Dignum, V., and Villadsen, J. (2014). “The AORTA architecture: integrating organizational reasoning in *Jason*”. In: *Engineering Multi-Agent Systems*. Ed. by Dalpiaz, F. et al. Vol. 8758. Lecture Notes in Computer Science. Springer International Publishing, pp. 127–145.
- Jensen, A. S., Dignum, V., and Villadsen, J. (2015a). “A framework for organization-aware agents”. Submitted to the journal of Autonomous Agents and Multi-Agent Systems.
- Jensen, A. S. (2015). “Model checking AORTA: verification of organization-aware agents”. In: *ArXiv e-prints* 1503.05317.

Acknowledgments. The results presented in this thesis could not have happened without the help and support I received from colleagues, friends and family.

I am grateful for the support, advice, help and dedication from my supervisor, Jørgen Villadsen. Furthermore, I thank Virginia Dignum for making it possible to visit TU Delft, for many interesting and fruitful discussions and for the collaboration on several of the results presented in this thesis.

I thank the section for Algorithms, Logic and Graphs for providing an excellent social and academic environment.

Finally, I am truly indebted to Eva for putting up with the inevitable frustration that comes from a PhD study, for trying to understand what I was going on about and for her support during my travels (even when it meant taking care of our two daughters alone).

Kongens Lyngby, November 18, 2015

Andreas Schmidt Jensen

Contents

x

CHAPTER 1

Introduction

Imagine entering an auction house to buy a lamp. You find out how to participate in the auction, and decide to bid on a particular lamp. After a short while, you are abruptly thrown out from the auction house without any explanation of what you did wrong. This scenario may seem unlikely: surely, there would be *some* indication of what you did wrong, or the employees at the auction house would take measures to prevent you from doing "wrong things" altogether. If we instead consider an *electronic* auction populated by *intelligent agents*, we argue that the scenario becomes more plausible[1]. For example, agents entering the electronic auction house would be required to follow certain rules, such as registering before bidding, or pay for items won at the auction. In general, we cannot assume that agents are able to understand and follow the rules in an arbitrary, complex environment. The goal of this work is to build intelligent agents that can reason about and act upon the rules and expectations from the system they participate in.

The term agent usually refers to an entity that can act on behalf of someone else. In the context of artificial intelligence, an (intelligent) agent is often thought of as "*an entity that functions continuously and autonomously in an environment in which other processes take place and other agents exist*" [Shoham, 1993]. Shoham proposed the agent paradigm as a novel way to create programs, based on the notions of agents and agency. The paradigm has led to the development of various agent design methodologies and programming languages, allowing programmers to create programs based on agency and autonomy.

Autonomy comes at a price: it becomes hard to predict the agent's behavior, especially in a system consisting of many agents. Furthermore, as indicated in the example, the system might define rules that agents are expected to comply with. To ensure compliance, it is often suggested to impose an *organization* on top of the system (similar to what is done in human societies). The organization defines the structure of the system (in terms of roles and their relations) and expectations (in terms of objectives, rules and norms) of the agents in the system. Rather than using this as a way to take away the agents' autonomy (as this would contradict the notion of agents), the aim is to find a balance between predictability and efficiency. It might be acceptable to break a few rules if it leads to a more efficient execution of the system.

The definition of an organization does not ensure stability and predictability in a system by itself. To operationalize the organization, the agents in the system must be able to reason about it. For example, when entering the auction house, the agent should be able to, e.g., figure out that it will eventually be banned if it bids on items it does not intend to pay for. Agents that can perform this kind of reasoning are usually called *organization-aware*.

[1] Though we have yet to define what is meant by an agent.

In several cases, the ability to reason about organizations has been successfully integrated into agent programming languages. However, in most cases, the organization (or at least the operationalization of it) is tailored to the specific language, making it hard to use in combination with other languages and platforms. This is paramount for open systems, such as an electronic auction house, in which different kinds of agents should be able to participate. In this thesis, we aim to show that it is possible to devise an organizational reasoning component that can be integrated into different kinds of agents, allowing them to reason about different kinds of organization.

The rest of the chapter is organized as follows. In Section 1.1, we motivate our research and present our main research goals. We present a case study of an electronic auction house in Section 1.2, which will serve as a running example throughout the thesis. Finally, we provide an overview of the thesis in Section 1.3, and relate the thesis with earlier published work in Section 1.4.

1.1 Motivation and Research Goals

Our motivation for this work is based on a desire to allow different kinds of intelligent agents to reason about and participate in the same organizations. To create organization-aware agents, we require three things:

1. An **organizational model** that defines what is expected of the agents.
2. An **agent programming language** that allows us to implement intelligent agents.
3. A bridge between the model and the language that enables the agents to **operationalize** the model.

The first and the second point have been thoroughly studied and has led to the development of several organizational models [Dignum, 2004; Esteva et al., 2002; Hübner et al., 2002] and agent programming languages [Bordini et al., 2007; Dastani, 2008; Hindriks, 2009]. However, while the third point has been investigated and has led to several promising results [Alechina et al., 2012; Esteva et al., 2004; Hübner et al., 2010; Hübner et al., 2006; Tinnemeier, 2011], most results are either tailored to specific organizational models or will only function with specific agent programming languages.

Most promising is perhaps the work of Hübner et al. [2010], who has created a general model for *organizational artifacts*, which can encapsulate potentially any organizational model in a number of artifacts that agents can interact with. However, different agents interact with the artifacts in different ways, and an approach suitable for one agent is not easily transferred to another agent (written in a different language). Furthermore, the organizational model is encapsulated in artifacts, which, depending on the model, must be interacted with differently.

This leads to our main research goal: to create a framework that fulfills the third requirement by means of a general bridge that (potentially) allows agents in any language to reason about organizations described by any organizational model. This might be an ambitious goal, but we believe it is paramount for the success of organization-enabled

systems (especially open systems) that agents of any kind are able to enter and successfully participate. From this, we have identified a number of sub-goals that (when fulfilled) contribute to our overall research goal:

1. To reason about different organizational models, we propose an **organizational metamodel**, which is able to capture common concepts from existing organizational models.
2. We require that our organizational reasoning is underpinned by a **formal operational semantics**. This should be done for two reasons. First, agent reasoning is traditionally founded in a formal setting (often modal logic) [Cohen and Levesque, 1990; Rao and Georgeff, 1991], which contributes to a rigid and profound understanding. Second, operational semantics match closely the implementation of an interpreter and is thus a natural step towards a practical framework.
3. We develop an **organization-oriented programming language** that allows individual agents to perform organizational reasoning based on their own state and preferences.
4. As a way to separate the organizational reasoning from the agent, we propose a **stand-alone component**, which comprises the metamodel, interpreter and programming language. By keeping it separate, it is independent from the agent and is more easily integrated with different kinds of agents.
5. To show that our component truly enables organizational reasoning, we perform **verification** of the system. This is a natural step forward from having a formal operational semantics, since the operational semantics make translating a program into a finite-state model straightforward.
6. Finally, we aim to show that the autonomy of the agents is not too severely affected by the component. This can be done by showing that the agents are able to **decide** between own desires and organizational goals.

In order to fulfill our research goals, we proceed as follows. First, we conduct a thorough review of the state of the art in the areas of organizational modeling (of agent societies) and agent programming languages. We then claim – based on our findings – that the current bridges between organizational models and programming languages are too specific and tailored to individual needs. Then, having convinced ourselves that our goal is sound, we proceed to work on each of the listed sub-goals.

1.2 Case Study

Throughout the thesis, we will refer to a running example, which serves two purposes: first, it will help in understanding the theory behind the AORTA reasoning framework and second, it can be executed in the implemented tool, showing the practical usefulness of our framework.

We consider agents participating in an electronic auction house (EAH). Participants in an auction bid openly against each other, with each bid being higher than the previous bid. The auction ends at a predetermined time. In the context of this thesis, the most interesting kinds of auctions are conducted *electronically* – for example at Internet auction sites such as eBay.

The idea of modeling auction houses in multi-agent systems is not new, but has been done by several people with various purposes [Dastani et al., 2005; Esteva et al., 2002; Noriega, 1997]. Here, the purpose is to present an example in which agents are required to understand the rules of the system in order to achieve their personal goals. That is, if the agents are unable to adhere to the rules of the system, they risk being thrown out. We use the example to show that our work indeed allows agents to understand and reason about organizations, and furthermore are able to make informed decisions about whether or not to follow the rules (with the proper sanctions taken into account).

> *Alice wants to sell her PH-lamp and has decided to use the electronic auction house* SellOut Inc. *to do so.* SellOut Inc. *requires users to register with a valid address and credit card information before bidding on or selling items. However, because the verification process is slow, the manager has decided that it should be possible to enter auctions with a registered but non-validated account.*
>
> *Alice registers and puts her lamp up for sale. Bob and Carol both wants to buy the lamp, so they register and starts to bid against each other. Meanwhile, the manager has completed the verification process and while Alice and Carol could be verified, Bob had provided an expired credit card. As a result, he is not allowed to bid on items in the auction house and must leave immediately.*
>
> *Finally, Carol wins the lamp and once she has paid for it, she can leave the auction.*

The example serves to show a number of interesting points. First, by not forcing agents to become verified before participating in an auction, we allow them to register with false information in order to try to circumvent the rules. Second, by having a manager taking care of the verification process and ensuring that participants follow the rules, we illustrate an important fact of organizations: they consist of *agents*. That is, an organization should not be thought of as an active entity, but rather a set of rules that the agents should follow. It is the responsibility of the agents to ensure not only that the rules are followed, but also that proper measures are taken when the rules are violated. We can thus show that certain agents are able to sanction other agents if they choose not to follow the rules. Third, the bidding process is largely non-deterministic and makes an interesting case for verification. For example, it allows us to verify that agents that bid with unverified information are always removed from the auctions, and that the agents can use the organizational structure to figure out how to become verified.

Finally, having a common example throughout the thesis lets us more easily explain and unify the parts of our work, since everything is put into the same context.

1.3 Overview

We provide an overview of the remainder of the thesis, which is divided into four parts:

Part I (Setting the Stage)
We provide the reader with a sufficient background for understanding the primary contributions of the thesis. In Chapter 2, we describe intelligent agents and multi-agent systems, and give an overview of the prominent Belief-Desire-Intention (BDI) model, which has laid foundation to more than a few agent programming languages. We furthermore describe the concept of multi-agent organizations, and discuss what it means for an intelligent agent to be *organization-aware*. In Chapter 3, we discuss the current state *organizational reasoning*. We investigate previous and current work on making agents organization-aware in order to identify methods, tools and shortcomings. We pose a number of requirements for successful reasoning about organizations, and use the findings in the rest of the thesis to provide sufficient evidence for the usefulness of AORTA.

Part II (Adding Organizational Reasoning to Agents)
We start by introducing the AORTA metamodel, which is based on the notions of roles, objectives and obligations, and the AORTA-component, which uses the metamodel as a basis for the reasoning (Chapter 4). We show how well-known organizational models can be translated to the metamodel. We provide operational semantics for the AORTA-component, making it possible to execute agents with organizational reasoning capabilities (Chapter 5).

This part fulfills subgoals 1 and 2 and partly fulfills subgoal 4.

Part III (Engineering Organization-Aware Agents)
We present the AORTA architecture, a Java-based implementation of the AORTA framework (Chapter 6). We then provide examples of organization-aware agents implemented in AORTA, integrated with existing agent platforms (Chapter 7). Finally, we show how we can use *model checking* to verify agents that use AORTA to perform organizational reasoning using Agent Java PathFinder (Chapter 8).

This part fulfills subgoals 3, 4 and 5.

Part IV (Deciding Between Influences)
We propose a consequence-based approach to decision-making that allows agents to consider the expected consequences of their actions (for example based on the expectations from the organization) before deciding what to do. We present our formal model, which is based on qualitative decision theory (Chapter 9), and we show a proof-of-concept implementation of the model, integrated with AORTA and the *Jason* agent programming language (Chapter 10).

This part fulfills subgoal 6.

Finally, we claim that by having fulfilled the subgoals, our main research goal is achieved. We do so by going through the list of requirements for organizational reasoning in Chap-

ter 11. We then conclude the thesis and suggest directions for future research in Chapter 12.

1.4 Relation with Earlier Published Work

The content of this thesis is based on research we have previously published in a number of papers. The work has been extended and integrated to form a coherent story. Below, we relate the chapters of the thesis and their relation with earlier published work.

Chapter 4 is based on our initial paper on AORTA [Jensen and Dignum, 2014] as well as [Jensen et al., 2015a], but is greatly extended to include a more thorough description of the semantics of norms.

Chapter 5 is based on [Jensen et al., 2015a], which presents the operational semantics based on obligations, but is extended to consider prohibitions as well.

Chapter 6 is based on [Jensen et al., 2014], which describes the implementation of an earlier version of the AORTA architecture. It is thus updated to conform with the operational semantics presented in Chapter 5.

Chapter 7 is based on [Jensen et al., 2014] as well, but is extended to include integrations with various other agent programming languages.

Chapter 8 is an extended version of [Jensen, 2015], which includes more detailed benchmark results and a step toward verification of certain key criteria of organizations, such as being "good" or "efficient".

Chapters 9 and 10 are based on [Jensen, 2013a], and the work is primarily extended with an implementation in Java that is integrated with AORTA and the *Jason* agent programming language.

PART ONE

SETTING THE STAGE

We provide the reader with a sufficient background for understanding the primary contributions of the thesis. In Chapter 2, we describe intelligent agents and multi-agent systems, and give an overview of the prominent Belief-Desire-Intention (BDI) model, which has laid foundation to more than a few agent programming languages. We furthermore describe the concept of multi-agent organizations, and discuss what it means for an intelligent agent to be *organization-aware*. In Chapter 3, we discuss the current state *organizational reasoning*. We investigate previous and current work on making agents organization-aware in order to identify methods, tools and shortcomings. We pose a number of requirements for successful reasoning about organizations, and use the findings in the rest of the thesis to provide sufficient evidence for the usefulness of AORTA.

CHAPTER 2

The Agent Paradigm

Our work on programming organization-aware agents is based on research in multi-agent systems and organizational modeling. For completeness, this chapter presents the key concepts used in these areas, such that our research is put in the correct context. Our main contribution is an organizational reasoning component, AORTA, which includes a programming language for specifying reasoning rules for individual organization-aware agents. We integrate the component into agents that are based on the "belief-desire-intention" (BDI) model, so a large part of the chapter is devoted to the concepts of this model. Furthermore, AORTA uses the concept of an organizational metamodel, designed to support the use of different established organizational models. We therefore present the general concepts of organizational models and briefly discuss some of the well-known models, as this will prove useful later.

2.1 Intelligent Agents

One of the overall goals of our work is to make clear the requirements for providing rational agents with organizational reasoning capabilities. Before we can do this, we must reach a common understanding of rational agents and of organizational reasoning. In this section, we present our understanding of agents and multi-agent systems and provide an overview of previous work on practical reasoning for agents and on how to program multi-agent systems. The purpose is therefore *not* to provide a comprehensive overview of multi-agent systems and the BDI architecture; for this, we refer to [Bordini et al., 2007; Bratman, 1987; Shoham, 1993; Wooldridge, 2009].

The term "agent" is originally used to denote someone acting on behalf of someone else. In artificial intelligence (AI), such an agent could correspond to software acting as a personal assistant for the user, for example, a voice assistant, like Apple's Siri; or an agent performing automatic stock trading. While such uses do resemble a kind of intelligence (e.g., voice recognition, intelligent predictions), when referring to agents, one usually refers to an entity able to perform a broader variety of tasks, is able to cooperate with other agents, and is in some sense *goal-directed*. We call such agents *intelligent agents*.

An intelligent agent is an entity that is situated in an environment in which it can *act* and *sense* [Russell and Norvig, 2010; Shoham, 1993]. Intelligent agents[1] are usually characterized by their ability be proactive, reactive, autonomous, and social [Wooldridge, 2009].

[1] In the following, we refer to intelligent agents as just agents.

- *Proactive behavior* enables agents to exhibit goal-directed behavior; the agent will actively perform actions that it believes brings it closer to the achievement of its goals.

- *Reactiveness* allows agents to adapt to changes in the environment; since the agent is situated in an environment, the ability detect and react to changes enables the agent to adjust its actions to the situation at hand.

- *Autonomous* agents can choose by themselves how to achieve their goals; if the agent has several ways of completing a task, the autonomy of the agent allows it to decide which way is better in a given situation.

- *Social* agents can communicate and cooperate with other agents, such that complex tasks are more easily solved.

An agent consists of a number of sensors and actuators. The sensors are able to perceive the feedback it receives from the environment and create percepts, which the agent is able to use in its reasoning. The reasoning results in actions that the actuators will perform, which will potentially change the environment.

2.1.1 Multi-agent systems

A multi-agent system (MAS) is then a system of agents in an environment. The agents in the system may be acquainted with some (or all) of the other agents in the system, and can communicate with them. A multi-agent system takes advantage of the abilities of each agent in order to complete tasks that are difficult (or impossible) for the individual agents to complete. In that sense, the sum (the multi-agent system) is greater than the parts (each agent).

The environment in which an agent is situated can be classified using four dimensions [Russell and Norvig, 2010]:

Accessible versus inaccessible: An accessible environment is an environment in which the agent will always have a complete and accurate picture of it. Most real-world environment are considered inaccessible.

Deterministic versus non-deterministic: When an agent performs an action in a deterministic environment, it is guaranteed that exactly this action will be performed. In a non-deterministic environment, this may not be the case.

Static versus dynamic: A static environment will only change when an agent executes an action. Dynamic environment may change even without interactions from an agent. A real world environment will usually be dynamic.

Discrete versus continuous: Discrete environments are characterized by a fixed number of possible actions and types of percepts. Continuous environment are more realistic, but will also increase the complexity of the system.

The EAH is an accessible, deterministic, dynamic and discrete environment: Every agent knows about the available auctions, the bids and other registered customers. To simplify, we assume that actions (if allowed) always succeed, though in real life, this might not be the case (e.g. if an attempt to bid at an auction fails due to bad network connection). The environment is dynamic in the sense that auctions have an end time at which point they automatically close. Finally, it is discrete, since there is a fixed number of actions and percepts.

2.1.2 Practical reasoning

From a philosophical point of view, practical reasoning is the use of *reason* to decide how to *act*. That is, practical reasoning is directed towards action. Contrast this with theoretical reasoning, which is directed towards beliefs. For example, reasoning about which car is better is theoretical reasoning, since it only changes my beliefs about the world, whereas reasoning about which car to buy is practical reasoning, since it, potentially, lets me perform an action (buying a car).

It should be clear that we want agents to perform practical reasoning; agents should reason about what to do – and then do it. Furthermore, agents that are situated in an environment usually have a purpose, and (hopefully) they have the means to achieve goals leading to this purpose (perhaps in cooperation with other agents). Since they are proactive, we can assume that they will actually attempt (by themselves) to achieve their goals. Before going into details about exactly *how* agents will achieve their goals, let us take a step back, and look at practical reasoning from the *human* perspective. That is, how does a human being perform practical reasoning, and how can this be formalized into an architecture for programming agents.

Practical reasoning can be divided into *deliberation* and *means-end reasoning*. Deliberation is the process of deciding what to do (or what state of affair to achieve); means-end reasoning is the process of deciding how to do it [Wooldridge, 2009]. In deciding what to do, it seems reasonable to assume available a (finite) list of things, one would like to do. These things – usually called *desires* – may be mutually inconsistent, e.g. I could desire to go to the movies and at the same time desire to go to the beach. Deliberation is then the process of choosing a desire and deciding to achieve it. The decision will probably be based on *beliefs* about the current state of affairs: if I believe it will rain today, I will be more likely to choose to go to the movies. On the other hand, if I believe I have another appointment today, I might not choose any of the desires. The deliberation phase ends when I have chosen a desire to achieve; I now have to decide *how* to achieve it. For example, if I decide to go to the movies, I have to plan how to get there, how to buy a ticket, where to sit, etc. Thus, the result of means-end reasoning is a *plan* for how to achieve a chosen desire.

The mental attitudes of beliefs and desires can thus be used as a way to explain human practical reasoning, and may therefore be a reasonable choice for modeling agents. It is however argued that a third mental attitude is required to fully capture practical reasoning, namely *intentions* [Bratman, 1987]. The notion of intention can be used to characterize actions: I am buying a movie ticket *intentionally*. Furthermore, it can be used to

characterize the mind: I *intend* this morning to go to the movies tonight. The difference between the two is in the temporal aspect: the former characterizes a present-directed intention, while the latter characterizes a future-directed intention. Bratman argues that the notion of intentions is usually associated with the future; a present-directed intention is merely embedded in the action currently performed. Thus, in the following when we say intentions, we refer to the future-directed kind. Intending to go to the movies tonight means performing actions that eventually leads to me being at a theater tonight watching a movie, but at the point in time where this intention was formed, I might not know exactly which actions to perform to get there. However, since I intend it, I am committed to figuring out which actions to take in order to achieve my goal. This includes looking up which movies are playing in what theaters, how to get to the theater (once one has been chosen), deciding how to buy the ticket (online, via phone or at the theater), and so on. As the day progresses, I get closer to achieving the goal by continuously planning what to do to complete my intention. An intention can thus be viewed as an incomplete plan with the holes being filled "as we go", since not everything can be accounted for initially. It is of course also possible that at some point, I decide to drop the intention because e.g. I feel sick, or I get a better offer.

It has been argued that intentions can be reduced to desires and beliefs, and that it is therefore not necessary to model intentions as a separate attitude. The idea is that an intention to achieve a state of affairs can be reduced to (1) a desire to achieve that state of affair and (2) a belief that the state of affair will eventually be achieved. This is however a simplification, which does not fully capture the idea of intentions. As argued by Cohen and Levesque [1990], once intending to do something, one is *committed* to do it and will not easily give up on the commitment (otherwise, was it really a commitment?). This cannot be fully captured by a simple "desire-belief" model, since it is not in the same way future-directed. As Bratman states, *"Why don't we just cross our bridges when we come to them? An adequate answer to this question must return to the central fact that we are planning creatures. We form future-directed intentions as parts of larger plans, plans which play characteristic roles in coordination and ongoing practical reasoning"* [Bratman, 1987, p. 8]. In other words, a desire to go to the movies tonight and a belief that I will be at a theater watching a movie tonight does not help me plan how to get there. Since we are "planning creatures", it is paramount to have a plan for how to achieve my desire in order to believe that I will be at the theater tonight, since otherwise, my belief has no support.

Much effort has been put into formalizing the belief-desire-intention (BDI) model, first in a purely logical setting (most notably Cohen and Levesque [1990] and Rao and Georgeff [1991], both of which capture the concepts in modal logic) and later as the BDI architecture [Rao and Georgeff, 1995], which has been the basis for many of the agent programming languages that are being used now. The BDI logics were used to prove certain properties of BDI agents. For example, Bratman argues that it is irrational for an agent to intend to perform an action, that the agent beliefs it will not do, thus it should hold that $\mathsf{INTEND}(\phi) \rightarrow \neg\mathsf{BEL}(\neg\phi)$, i.e., if the agent intends to do ϕ, it does not believe that it will not do it. The *asymmetry thesis* includes this principle along with other principles describing the relationship between beliefs and intentions, desires and intentions, and beliefs and desires. Rao and Georgeff [1998] furthermore examined several

multi-modal BDI systems to determine in which of them the principles hold. Later it was shown that several of the asymmetry thesis principles hold in AgentSpeak(L) [Bordini and Moreira, 2004], making a strong case for using agent programming languages with a logical foundation.

2.1.3 Programming agents

Bratman's "belief-desire-intention" model has been generally accepted in the agent community as an adequate model for practical reasoning agents. The approach usually taken is to enrich agents with *mental states* [Shoham, 1993], one for each of the mental attitudes. A BDI agent is thus an agent with the mental attitudes of beliefs, desires and intentions, representing respectively the information, motivational and deliberative states of the agent. In this subsection, we proceed to explain the concept of *agent-oriented programming* (AOP) and the *BDI software architecture*.

Shoham [1993] argued in his seminal paper on AOP that *"agenthood is in the mind of the programmer"*. That is, what makes a (software or hardware) component an agent is the fact that the programmer has chosen to analyze it in terms of the mental attitudes discussed above. Thus, according to Shoham, a robot vacuum cleaner is considered an agent, once we (the programmers) decide to enrich it with beliefs, desires and intentions. The difference should be clear: a *robot* vacuum cleaner is simply programmed to move between locations, cleaning the floor on its way. Variables indicate its current location and functions move the agent to new locations, orders it to clean, etc. An *agent* vacuum cleaner holds *beliefs* about its current location, *desires* to clean its environment and, e.g., *intends* to clean the room in which it is currently situated.

Of course, analyzing is not enough; we need appropriate tools for programming such agents using beliefs, desires, and intentions. Shoham proposed the AOP paradigm as a specialization of object-oriented programming (OOP). His goal is clear: an AOP system should consist of a language for the mental state with clear syntax and semantics and an interpreted programming language in which agents are defined, with semantics faithful to the theory of the mental state. In his paper, Shoham proposed a simple agent programming language, AGENT-0, and an interpreter for it. Since then, a variety of agent programming languages, inspired by the BDI model, have been proposed. Examples include AgentSpeak(L) [Rao, 1996], *Jason* (based on AgentSpeak(L)) [Bordini et al., 2007], GOAL [Hindriks, 2009; Hindriks et al., 2001], 2APL [Dastani, 2008], 3APL [Hindriks et al., 1999], SAAPL [Winikoff, 2007], and Jadex [Pokahr et al., 2005]. Except Jadex, all of the above are founded in an operational semantics [Plotkin, 1981], explaining in a precise manner the behavior of the programming constructs of the languages. This have allowed researchers to investigate the relationship of properties of the languages with well-known properties of BDI logics [Bordini et al., 2004a; Jongmans et al., 2010]. We return to the problem of investigating properties in agent programs in Chapter 8 in relation to organization-aware agents.

Programming the agents is, however, not enough. To execute an agent, we have to look at the phases of practical reasoning, namely deliberation and means-end reasoning. We have briefly discussed each of the phases: deciding what to do, and then how to do

Algorithm 2.1 BDI interpreter

```
initialize-state()
loop
    options ← option-generator(event-queue)
    selected-options ← deliberate(options)
    update-intentions(selected-options)
    execute()
    get-new-external-event()
    drop-successful-attitudes()
    drop-impossible-attitudes()
end loop
```

it. A question that springs to mind is then this: How often should one perform these phases? As we now know, an intention is a kind of commitment, so it can be argued that once an agent has decided what to do, it should stay in the second phase, means-end reasoning, until it has brought about the sought state of affair. This describes one of three commitment strategies suggested by Rao and Georgeff [Rao and Georgeff, 1991], *blind commitment*. This is a very strong commitment, since the agent has no reason to drop the commitment, unless it has achieved it. The second, *single-minded commitment* is similar, but allows the agent to drop the commitment, once it believes it is no longer possible to achieve it. Finally, *open-minded commitment* means that the agent will be committed to *maintain* an intention until it has been completed or is no longer a goal.

To accommodate this, Rao and Georgeff [1995] proposed the BDI software architecture, based on Bratman's BDI model and their own formalization of it. The simple, abstract BDI interpreter is shown in Algorithm 2.1. The interpreter works as follows. An event queue holds external events (from the environment) and internal events (e.g., adopting a new goal). First, a number of options are generated based on the event queue and the agent's mental attitudes. From these options, a number of options are selected for adoption (the deliberation phase). The agent's intentions are updated with these new options, and the agent executes an atomic action (if there is an intention to do so at that point in time). New external events are added to the event queue, and the agent updates its desires and intentions by removing those that are successfully achieved or deemed impossible to achieve. The commitment strategy can be determined by these *drop* functions. For example, blind commitment can be obtained by never deciding that an intention is impossible to achieve.

Their interpreter is based on a logically closed set of beliefs, desires and intentions, and is thus not as such useful for practical purposes. To accommodate this, they make a number of choices of representation:

- The agent holds beliefs about the current state of the world, represented as ground sets of literals without disjunction and implication.
- Plans are introduced, as a way to represent means of achieving future states, consisting of a triggering event (an event that can invoke the plan), a precondition

(conditions under which the plan can be chosen) and a body (the primitive actions and subgoals that should be executed in order to complete the plan successfully).

- Intentions are represented using a "run-time stack of hierarchically related plans", and it is possible to commit to multiple intentions that can run in parallel, sequentially or using a suspension mechanism that can pause intentions until certain conditions hold.

Still not quite satisfied with the abstract BDI architecture, Rao [1996] proposed AgentSpeak(L), an agent-oriented programming language based on a restricted first-order language with events and actions. Instead of specifying the agent's behavior using modal formulas, it is determined by programs written in AgentSpeak(L). The mental attitudes can be ascribed to the agents by the designer. For example, the current state of the agent, its knowledge about the environment and other agents can be viewed as its current belief state. States the agent wants to bring about can be viewed as its desires, and programs (i.e., plans) that satisfy the desires can be viewed as the intentions. Rao notes that *"this shift in perspective of taking a simple specification language as the execution model of an agent and then ascribing the mental attitudes of beliefs, desires, and intentions, from an external viewpoint is likely to have a better chance of unifying theory and practice"* [Rao, 1996, p. 43]. Furthermore, we believe that this perspective makes agent-based languages more attractive to programmers, since a rather deep mathematical background is no longer required to design agents.

The *Jason* APL is an AgentSpeak(L) based APL, which is often used in research in connection with (among other things) artifacts [Omicini et al., 2008], organizations [Hübner et al., 2006], semantic web [Klapiscak and Bordini, 2009], and learning [González-Alarcón et al., 2013]. Thus, *Jason* has a large community and is actively developed and extended. In fact, it was designed with extensibility in mind, making it an obvious first choice for integration with our framework. We provide a short introduction to programming agents in *Jason* in Chapter 7, where we also present our integration.

2.1.4 Agents and artifacts

As we will see in the next section, there has been a shift in focus from issues concerning the individual agent to issues concerning a society of agents (when looking at organizational and coordination issues). However, before we go into details with the organizational perspective, we briefly comment on another proposal for organizing and coordinating agents. When focusing on each agent as an individual entity, tasks that require coordination and organization become increasingly complex concerning communication protocols or reasoning capabilities. Furthermore, it might not be suitable to model every entity and abstraction in a MAS as an agent. Even though most entities can be wrapped as agents, this will often result in a negative impact on the performance, if, e.g., the purpose of the entity is not goal-directed, or its behavior is largely determined by other entities.

Ricci et al. [2006] propose a solution using *artifacts*, which can be considered a first-class abstraction used to build the aspects of a MAS for which the agent metaphor is not suitable. Artifacts (as proposed by Ricci et al.) are generic objects that co-exist with agents

in an environment, characterized by a *reactive* behavior meaning that agents execute operations on the them. They are useful for, among other things, *gaining capabilities*, *coordination* and *organization*. The work by Ricci et al. has led to the development of the agents and artifacts (A&A) metamodel [Omicini et al., 2008], which defines an artifact as an object in the environment having the following properties:

- The **function description** defines the intended purpose of the artifact.
- The **operating instruction** describes the how the artifact should be used in order to realize its purpose.
- The **usage interface** provides details about the observable structure of the artifact.

The A&A metamodel has been used in various ways by researchers. Often used as an abstraction for describing coordination artifacts [Omicini et al., 2004], it has most prominently been embodied by the CArtAgO infrastructure [Ricci et al., 2007].

The CArtAgO infrastructure is a framework for developing artifact-based environments for MAS. It fully encapsulates the A&A metamodel and it has been integrated with both *Jason* and 2APL. BDI-agents from these platforms can enter environments developed using CArtAgO, thereby gaining access to the artifacts. The interaction with the artifacts work by means of external actions, and observable properties can be perceived by the agents. Artifacts have also been used with success for organizing agents [Hübner et al., 2010] by encapsulating an organizational model in a number of organizational artifacts. We briefly return to organizational artifacts in Chapter 3, where we discuss their use in relation to understanding organizational specifications.

2.2 Organizing Agents

In the previous section, we described intelligent agents and provided a brief history of the BDI model, which helped developing a theory for rational software agents and an architecture, whose principles has led to the development of several mature APLs. Our component extends intelligent agents with organizational reasoning capabilities. In this section, we introduce the key concepts of organizational modeling, so that sufficient background for the main contributions of the thesis will be established.

Recently, there has been a focus on *open agent systems* in which agents are free to enter and exit at will. Examples include electronic auction houses [Noriega, 1997], the domain of traveling [Dignum et al., 2008], and the Internet. In reality, any system that allows agents of unknown origin to enter can be considered open. These systems are inherently exposed to problems from different sources: objectives not being completed, deceitful agents trying to disturb the system, honest agents making mistakes (performing illegal actions, staying in restricted parts of the system, etc.), coordination gone wrong, or agents not living up to their responsibilities. This problem is usually solved by imposing an organization on the system (similarly to what is done in human societies). The idea of using organizations as a way to exercise control over agents in MASs is not new [Ferber et al., 2003], but has become increasingly recognized within the MAS community [Dignum,

2009], as can be seen by the growing number of publications and workshops on the subject (e.g., the COIN[2] workshop series [Balke et al., 2014]).

An organization provides by itself only *structure*: roles are specified and are related to different expectations according to the requirements or goals of the organization. As such, it provides no central control over the agents and no way to ensure the fulfillment of the expectations. To complicate things further, agents joining the organization may be designed by distinct designers, which altogether lead to uncertainty of the agents' motives and actions.

In order to manage the agents entering an organization, there is usually a set of control mechanisms available. The type of control exercised in an organization is often divided into two categories: *regimentation* and *regulation* [Jones and Sergot, 1993]. Regimented systems are characterized by the fact that certain actions are restricted, whereas regulated systems puts no such constraints on actions, but rather makes an attempt to ensure agents will not violate the constrains, despite being able to do so. This is often done by imposing sanctions on violating agents [Boissier and Riemsdijk, 2013; Riemsdijk et al., 2009]. In reality, the control mechanisms are often designed such that certain parts are regimented, while others are regulated. Consider, e.g., the EAH scenario: it might be impossible to bid on an item before being registered as a customer (regimentation), and while it is possible to bid without being verified, it is not allowed, so doing so may lead to sanctions (regulation).

Regulation is often put in effect by the *agents* in the organization. That is, the organization is not considered an active entity, but defines expectations (and it may be able to detect violations). It is then the agents' responsibility act upon other agents' violations of norms, for example, by removing them from the organization, deciding not to cooperate with them, etc.

Whereas strong guarantees can be made about regimented systems (e.g., a forbidden action is never successfully executed), the same is harder for regulated systems. It might be possible to guarantee that violation of certain expectations is always detected (especially if detection is done by the organization), but given the uncertainty of the agents in the organization it is not possible to guarantee that 1) a sanction is put in effect, 2) the sanction recovers the system and 3) the behavior of agent is affected by the sanction.

Since open systems do not in general pose restrictions upon the agents entering, it seems that in order to ensure that nothing undesirable happens, two options are possible: (1) make no assumptions about the agents' organizational reasoning capabilities and make everything regimented, or (2) assume that the agents are able to reason about the expectations of the organization and prefer sanctions to regimentation. The first option limits the actions the agent can take and ensures that nothing bad happens, but since the agents are not free to do as they like (since some actions are restricted), the execution of the system may not be very efficient. The second option lets the agents do what they want (within reason), but they may be sanctioned if they violate the expectations. However, if agents that do not understand the organizational model enter the system, they will first

[2]COIN stands for International Workshop Series on Coordination, Organization, Institutions and Norms in MAS.

of all not be able to understand what is expected of them, but secondly, if (and when) they violate the expectations, there is a risk that they do not understand the sanctions, which may then have no effect.

We look at organizations from the agent's perspective: in open systems, anyone can participate, but the success of the individual agent's participation depends on its ability to understand and reason about the system – including an organization, if present. We want any agent to be able to participate *successfully* in open systems. Successful participation is a vague statement, but our goal is clear: provide previously "organization-ignorant" agents with capabilities that allow them to perform organizational reasoning, such that they understand the organizational model of the system, can reason about participation and are able to choose whether to comply with the expectations that stem from the role(s) they enact. Such capabilities will essentially make agents *organization-aware* [Riemsdijk et al., 2009].

2.2.1 Modeling organizations

We have made clear the overall requirements for organizing agents, but we still have to describe a model for doing so. Put simply, such an organizational model should (1) *specify* the organization, and (2) make it possible to *operationalize* it.

1. The **specification** of an organization should describe the basic constructs of the organization (e.g., roles and objectives), the relation between these constructs (e.g., role dependency relations) and the constraints put on participants. We discuss the specification of organizations in the remainder of this chapter.
2. The **operationalization** of an organization depends on the agents participating [Riemsdijk et al., 2009], and is thus not directly present in the model; the agents' ability to reason about the organization is crucial for its success. Since organizational reasoning is a key aspect in our work, Chapter 3 is devoted to identifying the requirements of organizational reasoning.

Dignum and Dignum [2012] suggests the use of concepts from Organizational Theory (OT), which, even though it lacks formality, has been studied for years and has used in practice for many years with success. OT defines an organization as an entity in which individuals have *roles*, and use these roles to accomplish *collective goals*. Organizations are furthermore defined with a *purpose*; individuals in the organization have intentions and objectives, which lead to the overall collective goals.

Ferber et al. [2003] proposed three principles to keep in mind when designing so-called organization-centered MAS (OCMAS for short):

1. The organizational level describes the "what" and not the "how".
2. No agent description and therefore no mental issues at the organization level.
3. An organization provides a way for partitioning a system, and each partition (or group) constitutes a context of interaction for agents.

The first principle ensures that the designer follows the principle of *separation of concerns*. Agents are entities with certain capabilities (the "how") and a goal-directed behavior, but no concerns about what to do regarding the organization. Some sort of mapping from the specification of the organization to the functional level of the agent is then required for the agent to eventually achieve the organizational objectives (the "what").

The second principle allows the designer to abstract away from specific agents: the organization does not require a specific agent with the capabilities to achieve X and Y, even though the capabilities X and Y are required for the success of the organization. Of course, a mapping between the required capabilities and the agents in the organization is required, usually accommodated using roles, which are defined by the goals they are expected to achieve (i.e., the capabilities required to fulfill the expectations of the role).

Finally, the third principle involves partitioning the system into parts that are interdependent or with roles that could benefit from cooperation. This is often modeled using the notion of groups, e.g. by defining a group via the roles that are allowed to participate, or via collective objectives that the group is responsible for completing. An orthogonal method is the use dependency relations, which, e.g., puts the agents into a hierarchy, making it possible to delegate tasks to subordinates.

Furthermore, organizational models often take a normative view of a system, specifying how the agents ideally should behave, what they are allowed to do, and what is explicitly forbidden. Such obligations, permissions and prohibitions – collectively referred to as norms – affect the agents' behavior, usually via regulations, making them possible to violate. This can prove useful in situations where certain goals are more efficiently solved by violating a norm or when a personal goal is deemed more important than adhering to the norm [Dignum et al., 2002].

Various organizational models have been proposed, with the common purpose of adding structure to a MAS using the notion of organizational concepts. The models use concepts from OT, especially the notion of *roles*, abstracting implementation details away from expectations, and *objectives*, defining the desired outcome of the organization. We briefly summarize some of the most well-known organizational models:

- **Agent/Group/Role (AGR)** [Ferber et al., 2003]: The AGR model is based on *agents* playing *roles* in *groups*. An agent is defined as an active, communicating entity, but with no constraints on its mental capabilities. Groups are defined as a set of agents with some common characteristics. It is used for coordination of activities and for partitioning the organization. Finally, the role defines the functional position of an agent in a group, but in an abstract way that allows multiple agents to enact the same role.

  A number of axioms constrain the agents in the AGR model. For example, every agent is member of (at least one) group, and every agent enacts (at least one) role in a group. Furthermore, it is possible to define other structural constraints, for example a correspondence constraint, stating that agents enacting one role automatically enact another role as well.

- **ISLANDER** [Esteva et al., 2002]: ISLANDER is part of the Electronic Institutions Development Environment (EIDE), which focuses on electronic institutions (EIs).

EIs specified in ISLANDER are based on dialogs; agents play *roles* in *scenes* in which they participate in interaction protocols to fulfill their goals (e.g., agents in an auction market use an interaction protocol for bidding on goods, where certain criteria decide whether the agent has bid on an item). Scenes are defined in the performative structure, which is a collection of scenes that agents can navigate between. Each scene captures the interaction between agents by using well-defined communication protocols.

- **MOISE$^+$** [Hübner et al., 2002]: The MOISE$^+$ model is based on three organizational *dimensions*: the structural, functional and deontic dimensions. The structural dimension defines roles, links between roles, and groups. The functional dimension describes how the system should achieve its organizational goals by decomposing them into plans and distributing them to the agents as missions. Finally, the deontic dimension describes for each role, the missions they are permitted and obligated to commit to.
- **OperA** [Dignum, 2004]: The OperA model is a highly expressive model with a logical foundation, designed such that agents' desires and goals are distinguishable from the organizational aims. Agents are autonomous entities with personal goals and the characteristics of the organization are not dependent on the agents' desires. As such, the OperA model provides means for a designer to first define the agents and the organization independently and then create the links between them. The link between agents and the organization is made via *roles*. The roles describe the organization's view of the individuals whereas the agent describes each individual's own view.

Of course, there have been proposed various other models and methodologies for building organizational models for MAS (e.g. Gaia [Zambonelli et al., 2005], ROADMAP [Juan et al., 2002], and INGENIAS [Pavón et al., 2005]), but we found the models above most relevant for our work because of their shared characteristics and means for operationalization (see next chapter).

2.2.1.1 The OperA model

We conclude this chapter by demonstrating how the EAH can be specified in an organizational model. For this purpose, we use the OperA organizational model, which in some ways closely resembles our metamodel. Furthermore, we claim that subsets of the OperA model and the MOISE$^+$ model can be translated to the AORTA metamodel (see Appendix B). We do not provide a complete OperA model here, but rather use the scenario to show the possibilities of the model.

An OperA model consists of three parts: an organizational model (OM), a social model (SM), and an interaction model (IM). The OM describes the organizational structure and objectives in terms of groups, roles, norms and scenes. The SM describes how agents join the organization by enacting roles, and by agreeing on social contracts that provide the expectations on the behavior of the agents. The IM specifies the interaction agreement between the agents in the organization in terms of interaction contracts.

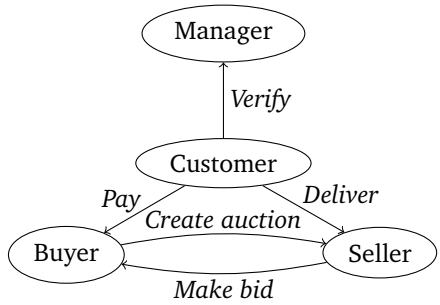


Figure 2.1: Social structure for the EAH scenario.

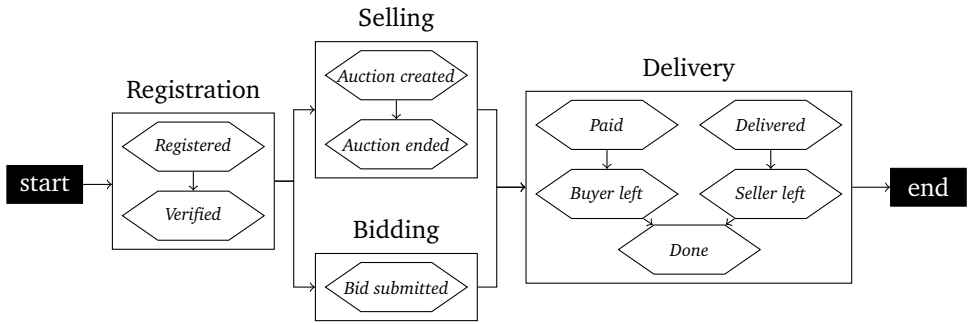


Figure 2.2: Interaction structure for the EAH scenario.

We focus on the OM (since the SM and IM are concerned with operationalization), which consists of four *structures*: the social, interaction, normative and communicative structures.

Social structure The social structure specifies the roles in the organization, along with their objectives, rights, norms and the dependency between roles. Roles are used to describe the social activities needed to achieve the organizational aims by abstracting away from the individuals that are supposed to achieve them. Objectives are refined by sub-objectives and the distribution of objectives is defined by a role hierarchy, which describes how roles depend on each other for the completion of objectives. A dependency relation of objective o is written $r_1 \succeq_o r_2$.

The social structure of the auction house is shown in Figure 2.1. For example, agents enacting the customer role depends on an agent enacting the manager role for verifying their credentials, written $customer \succeq_{verify} manager$.

Interaction structure The interaction structure for the auction house is shown in Figure 2.2 and specifies how the objectives of the roles are achieved by the enacting agents.

The notion of scenes and scene scripts are introduced to model this interaction. A scene represents an activity, which follows an abstract scene script. A scene script contains the roles in the scene, the norms imposed on the interaction, and the desired results. The scenes are connected via scene transitions, which specify a partial ordering of the scenes (registration is before bidding), and the condition under which a scene can start or can end. Role evolution defines how roles evolve as agents enacting them move from scene to scene. A role r_1 evolves into another role r_2 when a scene ends and a certain condition holds. The model distinguishes between *necessary* (evolution *must* happen given the condition) and *sufficient* evolutions (evolution *can* happen given the condition). For example, an agent enacting the customer role is allowed to evolve into the seller role when a transition from the registration scene to the selling scene occurs. A pair of roles is in conflict, denoted $r_1 \otimes r_2$, if they cannot be enacted by the same agent simultaneously.

Each scene has interaction patterns that define how each of the results should be achieved. An interaction pattern is a partial ordering of the objectives of the scene, providing the agents with information about the order in which the objectives are expected to be achieved. For example, to successfully complete the registration scene, the agent should first register, and then become verified (by a manager).

Normative structure The normative structure provides the agents participating in the organization with a way to *trust* each other. This is accomplished by introducing norms, which are expectations of the agents enacting a role. That is, by imposing norms upon the agents, other agents may assume that those norms are not violated. In OperA, there is a distinction between role norms (rules of behavior for enacting a role), scene norms (rules of behavior for agents in a scene) and transition norms (limitations to agents following a transition between scenes). For example, a role norm for a bidder would be to pay for an item that was won.

Communicative structure When participating in an organization (or any open system) it is important that the participants *understand* each other, i.e., they need a common ontology. The aim of the communicative structure is to describe communication primitives, such that the agents can communicate using a common ontology. The structure thus provides both the language for communication and the contents (or ontology). For example, there need to be a common definition of an *auction*, a *bid*, how to *register*, and so on.

2.3 Concluding Remarks

We provided an overview of multi-agent systems, intelligent agents and modeling of agent organizations. We have discussed how intelligent agents use practical reasoning to decide what to do, and how this principle can be used for programming such agents using a new paradigm, agent-oriented programming. Furthermore, we have provided the sufficient background for organizational modeling to continue discussing how intelligent agents can *reason* about organizations. Finally, we have provided a number of examples of existing organizational models that have been successfully used for organizing agents.

In the next chapter, we focus on organizations from the agent's perspective: we discuss how the agent can understand and reason about an organization to participate successfully in it.

CHAPTER 3

Organizational Reasoning

In the previous chapter, we introduced intelligent agents, the BDI architecture and discussed how multi-agent systems can be regulated using organizations. We proceed now with discussing one of the key requirements for our framework: how to add organizational reasoning capabilities to agents. This chapter investigates organizational reasoning; what it comprises and what agents can achieve with it.

Agents that are able to perform organizational reasoning are called organization-aware agents. Organizational reasoning as a concept covers many aspects: reasoning about entering and exiting an organization, reasoning about which roles to enact, whether to comply or violate certain norms and how to coordinate tasks with other members of the organization. In the following, we define what we mean by organizational reasoning and compare our approach to related work on organizational reasoning. We begin by looking at the three dimensions of organizational reasoning identified by Riemsdijk et al. [2009]:

- **Direction of organizational reasoning**: Top-down or bottom-up. Top-down reasoning starts from an organizational specification, whereas bottom-up reasoning lets the agents figure out among themselves how to organize. Though both approaches can co-exist in the same system, AORTA bases reasoning on an organizational model, and thus enables top-down organizational reasoning.
- **Understanding organizational specifications**: Investigating this dimension is required in order to be able to participate successfully in a top-down organization. The focus of AORTA is specifically on the second dimension: to provide agents with the possibility to understand organizational specifications.
- **Phases of participation**: By understanding the organizational specification, it is possible for the agents the move through the phases of participation. However, in order to successfully do so, the phases must be thoroughly investigated in order to identify the reasoning capabilities required in each phase. By employing organizational reasoning rules (discussed in Chapter 4), the AORTA framework allows agents to participate in each phase.

The chapter is divided into three sections, each considering different parts of organizational reasoning. In each part, we provide an overview of what has already been done in order to identify shortcomings in current approaches, and to "stand on the shoulders of giants". We furthermore briefly describe how we plan to accommodate these shortcomings in AORTA. In Section 3.1, we discuss how agents can understand the organizational specification. Section 3.2 goes through the phases of participation, and in Section 3.3 we focus on how normative aspects of organizations can be modeled.

3.1 Understanding Organizational Specifications

In order to understand an organizational specification, two things are required. First, the agent should understand the concepts used to describe the organization (i.e., how is a role specified, what are the objectives, etc.), and second, the agent should be able to operationalize the specification based on this knowledge. That is, given that the agent understands the notion of a role, how does it decide to enact that role, and how does the agent commit to completing organizational objectives alongside its own, personal objectives.

Much effort has been put into adding such capabilities to agents [Broersen et al., 2001; Castelfranchi et al., 2000; Dignum et al., 2000; Dignum et al., 2002; Hübner et al., 2007; Meneguzzi and Luck, 2009], and the focus has usually been on agents that are based on the belief-desire-intention (BDI) model. We divide the work into three categories: modified BDI architectures, middleware-based approaches, and component-based approaches.

3.1.1 Modified BDI architectures

Broersen et al. [2001] considers conflicts between beliefs, obligations, intentions and desires with a focus on a distinction between *internal* conflicts, e.g. contradictory beliefs and conflicting obligations, and *external* conflicts, such as a desire which is in conflict with an obligation. A simple way to handle conflicts is to impose an ordering between the concepts, such that, e.g., beliefs overrule desires or obligations overrule intentions. Depending on the ordering, different agent types emerge: if desires overrule beliefs, the agent is a wishful-thinking agent, while an agent that lets obligations overrule desires and intentions is social. The solution proposed, the BOID architecture, imposes a strict ordering between beliefs, obligations, intentions and desires, such that the order of derivation determines the agent's attitude. Each concept is mapped to a component in the architecture, which uses logical formulas to compute *extensions*, i.e., updates to the agent's mental attitudes. The architecture is designed such that the order of the components determines the agent's attitude, and each extension computed by a component is fed into the next component. The final component of the architecture is a planning component, which, based on the final extension, decides which actions should be performed by the agent.

A similar modification is the B-DOING architecture by Dignum et al. [2000; 2002], which represents obligations using Prohairetic Deontic Logic [Torre and Tan, 1999], a preference-based dyadic deontic logic which allows for contrary-to-duty obligations (i.e., obligations holding in a sub-ideal context). They propose a modified BDI-interpreter in which selected events are augmented with potential deontic events, which, put simply, are obligations and norms that may become applicable when choosing a plan. For example, the obligation to pay for an item at an auction is only applicable once the agent has bought an item, but by knowing that this *potentially* can happen, the agent can choose whether to adopt a plan to buy that item. Similar to the work done by Broersen et al. [2001], the concepts are given preferences in order to resolve conflicts. They furthermore map the orderings into a common scale, such that, e.g., very desirable situations could outweigh

the cost of violating certain obligations. One problem with these preference orderings is that they, in general, are static, meaning that each certain preference is given a weight, which is then used in all situations concerning that preference. Conflict resolution of mental attitudes is not a part of the core of AORTA, but we return to conflict resolution and making decisions based on preferences in Chapter 9 to investigate how it might be integrated.

Meneguzzi and Luck [2009] describe an approach that can modify an agent's behavior based on norms. While this approach does not modify the BDI architecture, it works by modifying the agent's plan library and intentions to adopt plans to achieve an obligatory state (and commit to those plans), or suppress (and drop) plans that could lead to a prohibited state. Their implementation is based on AgentSpeak(L), but is, according to the authors, sufficiently generic to be extended into any traditional BDI-style agent language. The AORTA framework incorporates behavior modification in a somewhat similar manner, in that it is possible to commit to or drop intentions to reach states described by norms. Since AORTA does not use the concept of a plan library, it is not possible to add or remove plans in the cognitive agent's plan library, but on the other hand it is designed to be generic enough to be usable by different cognitive agents.

While these, and other approaches, do enable agents to reason about organizational matters, they do so by changing the architecture of the agents. By doing so, existing implementations of agents may not work as expected using such architecture, meaning that in order to impose organizational reasoning capabilities upon these agents, changes in seemingly unrelated matters may be necessary. Furthermore, we believe that by abstracting the cognitive reasoning away from the organizational reasoning, we benefit from the principle of separation of concerns, which as mentioned is a key point for using organizations in multi-agent systems: *what* should be done is separated from *how* it should be done.

3.1.2 Middleware-based approaches

The ideas of Ferber et al. [2003] are more closely related to approaches based on a middleware. The idea in these systems is to keep organization and agents separate, and let all communication between the two parts happen through the middleware. A common trait of these systems is that all organizational actions and communication happen through the middleware. That is, any cognitive reasoning concerning the organization can be done by the intelligent agent, but acting upon it (e.g. enacting a role) must be done through the middleware. This has the effect that the middleware is able to decide whether an organizational action succeeds (i.e., is executed), e.g., if a cardinality restriction is imposed on a certain role.

One such example is the middleware for the MOISE$^+$ model, S-MOISE$^+$ [Hübner et al., 2006; Hübner et al., 2007], which separates organized multi-agent systems into a system level and an agent level. The system level, S-MOISE$^+$, provides an interface (a middleware) between the agents and the organization using a special agent, the OrgManager, to change the organizational state, ensuring organizational consistency. At the agent level, the system is integrated with an agent platform. An example of this is J-MOISE$^+$, which

joins *Jason* and MOISE$^+$, by making organizational actions available to agents, such that they can reason about (and change, using the OrgManager) an organization.

The EIDE framework [Arcos et al., 2005] comes with an agent-based middleware called AMELI [Esteva et al., 2004] for the ISLANDER organizational model [Esteva et al., 2002]. EIDE is characterized by dealing with electronic institutions (EIs), which often take the perspective that norms *by design* cannot be violated. In that case, this naturally means that electronic institutions differ from other agent-based approaches in that some of the autonomy of the agents is lost, since they cannot perform actions that would lead to a violation (this is not always the case; e.g. Aldewereld et al. [2007] operationalize norms in EIDE by detecting violation of norms and implementing sanctioning mechanisms).

The agents using AMELI interact with the EI via a so-called *governor* using a predefined set of messages the governor will understand. These messages concern among other things entering the institution, moving to a scene, and saying something in a scene. The governor can then agree to process the message (e.g., by executing an action that can either succeed or fail) or refuse it. The governors of AMELI are similar to the middleware of S-MOISE$^+$; the control of the institution lays on the institution side. Our approach is to let the agents decide by themselves whether they can enter the institution or move to a different scene. Furthermore, if the agents want to utter something in a scene, they should be free to do so, even if it means unintentionally bidding for an item, or being kicked out.

Similar to the middleware approaches is the idea of using artifacts as a mediator between agents and organization. The ORA4MAS (Organizational Artifacts for Multi-Agent Systems) approach is proposed by Hübner et al. [2010] and is another attempt to build a bridge between an organization and the agents in it. It is a general approach suitable for different kinds of organizational models, but in [Hübner et al., 2010] MOISE$^+$ is used as organizational model. They use artifacts, which they claim bring the control back to the agents (as compared to using a middleware), since the agents can, via their autonomy, choose whether to interact with the organizational artifacts of the system. However, we argue that different agents (i.e., programmed using different languages) interact with the artifacts in different ways, and an approach suitable for one agent is not easily transferred to another agent. On top of that, the organizational model is encapsulated in artifacts, which, depending on the model, must be interacted with differently.

Furthermore, we believe that the ultimate way of letting agents participate in organizations without removing too much of their autonomy is to let them perform the organizational reasoning and act upon it themselves (i.e., not through some artifact, middleware or something else). By integrating AORTA in agents, they are provided with organizational reasoning capabilities, but are still able to, e.g., decide not to commit to certain organizational objectives.

3.1.3 Component-based approaches

Finally, agents may be induced with organizational reasoning capabilities using a dedicated component. Here, the idea is to separate organizational matters from personal matters in the reasoning process.

Castelfranchi et al. [2000] consider *deliberate normative agents*, which are characterized by having explicit knowledge about norms in the environment and can decide whether to obey these norms. Their motivation for taking a component-based approach is that norms should not "just" be a filter on possible goals or constraints on the decision process, since the agent would always obey the norms if possible in those cases. Instead, it should be *"a motivated 'conscious' separate decision"*. Their approach is based on cognitive agents having with the usual mental attitudes (e.g. the BDI agents), and is based on (and tailored to) the compositional generic agent model (GAM) [Brazier et al., 2000]. GAM consists of components that manage and maintain the agent's knowledge and beliefs about the environment and control the behavior of the agent. The extension proposed by Castelfranchi et al. adds a society maintenance component that takes input from the agent's other components and uses this to incorporate new possible intentions. However, the approach is tightly coupled with GAM, since actual reasoning about norms takes place in the control component, which is refined to handle this.

A generic architecture for organization-aware agents is suggested by Riemsdijk et al. [2009], in which the agent is divided into two layers: a cognitive layer, which provides more or less the same reasoning capabilities as seen in, e.g. BDI agents, and an organizational layer, which communicates with the organization and handles decision-making that has to do with the organization (e.g., reasoning about role enactment). Our approach is similar to this; agents are enriched with an organizational reasoning component, which performs organizational reasoning *separate from* the cognitive reasoning performed by the agent. Our framework is generic enough to be useful for many different kinds of cognitive agents, such that they gain organizational reasoning capabilities without making changes to the "normal" reasoning.

3.2 Phases of Participation

Following Riemsdijk et al. [2009], we consider three phases of agent participation in organizations: *entering the organization*, *playing roles* and *leaving the organization*.

3.2.1 Entering an organization

An agent entering a system with an organization has to consider several things before choosing to enact a role. In general, the agent has to decide whether it wants to enter the organization. This includes, first of all, considering *why* it should enter it. What can the agent gain from entering the organization, which capabilities are gained, and what resources will be accessible. Furthermore, what capabilities are lost, and what does the organization expect from the agent, once it enters. A considerable amount of work has been done in the area of deciding which role(s) to adopt in an organization. Dastani et al. [2003] consider role-assignment in open societies by comparing the agent's goals with those of the role. For example, a role is compatible with an agent if the objectives of the role are a subset of the goals of the agent, and the objectives can be achieved using the agent's plans. An agent and a role are consistent if the objectives and goals

are not in conflict. Agents can use this to decide how well they match a role, and what is gained (e.g., the role provides plans for achieving personal goals) or lost (a goal is in conflict with an objective) by enacting a given role. They furthermore make a distinction between different kinds of role enactment, such as social or selfish enactment, which describe how an agent gives priority to the objectives of the role and its own goals. This distinction allows the agents in the organization to more easily distinguish between selfish and social agents trying to enter the organization, and may be able to use this knowledge to prevent selfish agents from entering.

Dignum and Dignum [2004] further extend this with personal *motivation*, which is characterized by the personal enrichment achieved by enacting a role (e.g. if the role provides the agent with a plan to complete one if its own goals, meaning the agent profits from the society), and by the power received by enacting the role (e.g. if the role provides capabilities to better realize the agent's goals, or the agent is able to delegate tasks to other agents through its role). Of course, a role might also diminish the agent's power by taking away capabilities (through norms) or by adding new goals to be achieved.

The notion of power is investigated in more detail by Carabelea et al. [2005], who consider the relationship between the roles in an organization before deciding whether to participate. By distinguishing between what an agent can do, has access to, and is allowed to do, they define what it means to have power over resources. Then, by defining social dependencies, it is possible to define a power relation between agents in an organization. Before entering the organization, the agents can use this to reason about what (or who) they will have power over, and who (if any) will have power over them. E.g., in the EAH scenario, the manager has power over the customer, since the customer depends on the manager to be verified as a legit customer. Consider an agent that has to decide whether to join the organization in order to become a customer; the agent can reason that any agent enacting the manager role will be able to decide whether the agent can be verified. If the agent furthermore knows that the current manager holds a grudge and will never verify the agent, it can use this knowledge to decide not to enter the organization.

Finally, the agents may consider their own capabilities before choosing to enact a role. That is, what capabilities does the agent have, and what capabilities are required for successfully enacting a given role. In order to consider the agent's capabilities, we should first make clear what kinds of capabilities an agent can have. A straightforward definition of agent capability is the set of states the agent can achieve. This has been investigated in the context of BDI logic by Padgham and Lambrix [2005], who consider BDI systems with a plan library. Here, having a capability for ϕ means having at least one plan with the goal ϕ as its trigger. It is also possible to extend the notion of capability to actions the agent is able to perform, percepts the agent can perceive and expressions the agent can utter. Aldewereld et al. [2012] extend the OperA model to include role capabilities, and agents entering the organization have to match the capabilities in order to enter. This decision is done by a gatekeeper, which decides, based on what the agent tells the gatekeeper that it can do, whether the agent is allowed to participate. Riemsdijk et al. [2011] investigate how agents can reason about their own capabilities using reflection. Reflection is used in certain programming languages (e.g. Java) to let a program refer to itself at run-time, making it possible to perform modifications of the run-time behavior

based on the reflections. Reflection in agent programming languages is thus a mechanism that, at run-time, allows the agents to refer to themselves and use this to modify their behavior. Being able to reflect upon their own capabilities makes it easier for the agents to choose the roles they fit best. Combining this with the other mechanisms mentioned, agents are able to perform an informed decision about which role(s) to enact.

3.2.2 Playing roles in an organization

Once an agent has entered the organization, it has to reason about the organization's expectations based on the role(s) it enacts. That is, the organization expects the agent enacting a role to achieve the objectives of the role, so the agent has to reason about 1) how to complete the objectives, 2) how this relates to the completion of its own goals and 3) what happens if it, for some reason, decides to violate some of the expectations. There is, of course, some overlap of reasoning between this phase and the previous, since this kind of reasoning played a part in deciding to enact the role.

In order to reason about completing organizational objectives, the agent has to understand what is required to complete an objective, and whether it can do so itself or by delegating the objective to other agents. The first point is concerned with relating the objectives of the organization to its own goals. The second point requires the agent to understand the organizational model, such that it can deduce hierarchical relations between different roles, and use this to decide how objectives can be delegated.

The organizational expectations might not be met by the agent, and in such case, the agent should know that. Depending on the agent's ability, different levels of understanding the model emerge: it might be able to understand before executing an action that this will violate an obligation, or it might just be able to understand that an obligation has now been violated. Furthermore, the agent should ideally be able to understand the sanctions that might be imposed upon it. If not, it has no way to decide whether to violate an obligation, since it does not know the impact a sanction may have. The B-DOING framework by Dignum et al. incorporates the possibility to reason about sanctions by introducing the *worth* of a state and *cost* of violating an obligation. This is used to make a preference ordering of obligations, such that agents can choose to violate obligations when the worth of the resulting state outranks the sanction. Meneguzzi and Luck [2009] lets the agents perceive norms in the environment, and then choose to accept or reject them. By rejecting, sanctions are imposed, thus reasoning about norms and sanctions in this case is done at different levels. First, when perceiving a norm, the agent decides to accept or reject a norm. Then, if it accepts the norm, it changes its behavior to conform to the norm. However, the change in behavior may not show until later, once the agent chooses to complete the objective described by the norm (as intentions to complete personal goals might still be chosen instead).

If the agent has no capabilities to complete a task, it should be able to find a solution to this in order to meet the expectations from the organization. Ideally, the agent should consider what to do in such situations before entering the organization, so it knows that it is able to complete the task somehow. As mentioned in the previous section, this can be done by considering the social power between the agents in the organization. A customer

in the auction house may be aware that it should be verified before buying or selling anything, but it is not capable of doing so itself. By considering the dependency relations, it can reason that agents enacting the manager role have social power over the agent in relation to the verification process, and thus the agent should ask the manager to complete the task.

3.2.3 Leaving an organization

Agents may have different reasons for wanting to leave an organization. Just as they had to decide whether to enter the organization, they have to decide whether they want to leave. These decisions are, of course, related. If, for example, an agent entered the organization in order to achieve a specific objective, it seems perfectly reasonable to want to leave the organization once that objective has been completed. Hormazábal et al. [2010] consider causes for dissolving organizations, and makes a distinction between necessary and sufficient causes. Necessary causes are causes that are required in order to dissolve the organization, but they are not sufficient, and require the members to agree in order for the dissolution to take place (e.g. due to inactivity by the agents). Sufficient causes are causes that, when detected, should automatically lead to the dissolution of the organization (e.g. when a deadline for the organization's lifespan is reached). Though their focus is on dissolution of the organization, it should be possible to transfer these ideas to the individual agents' decision of leaving the organization. For example, if other agents take no part in completing the organizational objectives, the agent may choose to leave because it reasons that there are not enough benefits from participating.

Once the agent has decided to leave the organization, it has to deact the roles it enacts in the organization. By deacting a role, the agent tells the organization to drop the expectations about the agent, but there might be situations where this is not ideal. For example, in an auction house, if the agent tries to deact the buyer role before having paid for the items it has bought, it should generally not be allowed to do so; at least not without getting sanctioned as well [Riemsdijk et al., 2009].

It might also be the case that the agent is thrown out of the organization (e.g., as a sanction), because it repeatedly violates obligations. This is only possible when enactment and deactment is determined by the whole community [Dastani et al., 2005], since other agents in the organization have to decide that the agent in question should be removed from the organization. For example, if an agent can only enact the seller role once he has been verified by the manager, the manager is able to determine enactment and deactment. If the agent then repeatedly violates its responsibilities as a seller (e.g., does not deliver the items it sells), it seems reasonable for the manager to ban (i.e., deact) the agent from the organization.

3.2.4 Summary

Having investigated the phases of participation, we briefly summarize the reasoning capabilities that can be expected of agents in an organization:

- **Entering an organization**
  - Which role(s) can the agent successfully play?
  - What power relations will be put in place when entering?
  - What capabilities are required?
  - What capabilities are gained?
  - Why should the agent enter the organization?
- **Playing roles in an organization**
  - What is required to complete a given objective?
  - How can the agent complete an objective without having the required capabilities?
  - How can the agents detect other agent's violations?
  - What are the consequences of violating a norm?
- **Leaving an organization**
  - When has the agent fulfilled its duties towards the organization?
  - What are the benefits of staying compared to leaving?
  - Can agents that perform repeated violations be removed?

We return to these items in Chapter 11, where we discuss to what extend our work contributes to providing agents with these capabilities. That is, whether agents using AORTA really are organization-aware.

3.3 Norms and Normative Multi-Agent Systems

The Oxford Dictionaries define a norm as *"a standard or pattern, especially of social behavior, that is typical or expected of a group"* [Oxford Dictionaries, Accessed online, Januar 9, 2015]. This definition very closely captures why norms are so appealing in the context of multi-agent systems. Since agents are autonomous, it can be hard to predict how each agent will act – especially in open societies. When taking the social aspect of intelligent agents into account, it thus seems very reasonable to investigate methods of presenting agents with the expectations that stem from the society.

A large part of the MAS community is devoted to researching norms and normative multi-agent systems. The term obligation is used already by [Shoham, 1993] in his seminal paper on agent-oriented programming, though here the focus is more on commitment than on norms. The bridge between law and agency has also been investigated with focus on intelligent agents [Conte et al., 1999], and lately the normative multi-agent systems have been discussed in detail at a series Schloss Dagstuhl seminars [Andrighetto et al., 2013]. The COIN series [Balke et al., 2014] is another example of a successful workshop series that focus on organizations and norms. There is thus no doubt that the idea of norms in multi-agent systems is relevant and useful, and our framework is certainly influenced by these ideas. In this section, we touch upon the formal work done on normative concepts, namely deontic logic, which studies the concepts obligations, prohibitions and

permissions. We furthermore provide a brief overview of the use of norms in various agent platforms to show how norms can be operationalized. These findings will prove useful in designing the AORTA framework in the next part of the thesis.

3.3.1 Deontic logic

In order to explain normative concepts in a rigorous and formal way, researchers have worked on capturing them in a formal system. Deontic logic is a symbolic logic that formally studies the concepts obligations, prohibitions and permissions. The first real deontic logic was created by Wright [1951], and it has now evolved into Standard Deontic Logic (SDL). SDL is a modal logic that provides modalities for obligations (O), prohibitions (F) and permissions (P). Given a language of propositional logic, SDL consists of the following axioms and rules:

$(Taut)$	All tautologies of the language
(K_O)	$O(p \rightarrow q) \rightarrow (Op \rightarrow Oq)$
(D_O)	$\neg O\bot$
(P)	$Pp \leftrightarrow \neg O\neg p$
(F)	$Fp \leftrightarrow O\neg p$
(MP)	$p, p \rightarrow q \vdash q$
(N_O)	$p \vdash Op$

From the axioms and rules, we can derive theorems of SDL:

$$Op \leftrightarrow \neg P\neg p \tag{3.1}$$

$$O(p \wedge q) \leftrightarrow Op \wedge Oq \tag{3.2}$$

$$O\neg p \rightarrow O(p \rightarrow q) \tag{3.3}$$

$$Op \rightarrow O(p \vee q) \tag{3.4}$$

Most of the theorems intuitively seem reasonable. For instance, Theorem 3.1 means that if one ought to do p, it is not permitted to not do p. However – and this is a problem of many variants of deontic logic – some theorems create paradoxes that create odd situations. The paradoxes often have something to do with how the formulas are read. For example, if we read $O(p \rightarrow q)$ (wrongly) as "if we do p, then we ought to do q", then by Theorem 3.3, if we refrain from not doing p (i.e., do p), we ought to do some q, for instance levitate. The last theorem is also known as Ross' paradox, which states that if we ought to do p, then we ought to do p or q. The example is as follows: "if we ought to mail the letter, then we ought to mail the letter or burn it", which was probably not the intention. Because of these and other paradoxes, much work in deontic logic is concerned with proposing alternative logics that can account for the paradoxes.

One important problem in deontic logic is the problem of expressing conditional obligations, i.e., obligations that should only hold given certain conditions. For example, how does one express "if I have bought an item at the auction, I ought to pay for it"? A suggestion could be $O(bought \rightarrow pay)$, but as we saw from Theorem 3.3, this can be quite problematic. Another suggestion could be to express it as $bought \rightarrow O(pay)$, however this

results in another well-known paradox, Forrester's paradox, or the paradox of the gentle murder. The paradox states that it is forbidden to commit murder, but if you do commit a murder, you ought to do it gently. Assuming that we do commit a murder and that (not surprisingly) murdering someone gently implies murdering them, we have the following SDL theory:

1. $F(murder)$
2. $murder \rightarrow O(murdergently)$
3. $murdergently \rightarrow murder$
4. $murder$

We call the second obligation a contrary-to-duty obligation because $\neg murder$[1] and $murder$ are contradictory. The paradox arise because we can derive $O(murder)$, which is inconsistent with $F(murder)$:

5. $O(murdergently)$ $2, 4, MP$
6. $O(murdergently \rightarrow murder)$ $3, N_O$
7. $O(murdergently) \rightarrow O(murder)$ $6, K_O$
8. $O(murder)$ $5, 7, MP$

Even though the set of premises seems to be consistent, it is not the case, because SDL is not able to express contrary-to-duty obligations. Contrary-to-duty obligations are interesting in the MAS setting, since provides a natural way to specify sanctioning. A sanction is thus a contrary-to-duty obligation. For example, if a customer should be verified before selling an item (the obligation), but if the customer puts an item up for sale before being verified, he should take down the item again and leave the EAH (contrary-to-duty obligation). Prohairetic Deontic Logic (PDL) is proposed by Torre and Tan [1999] as a way to solve Forrester's paradox and other paradoxes. The semantics of PDL is based on a deontic preference ordering, which orders worlds in terms of ideality. They show that the ideal world is one where murder is not committed, and that the theory is consistent. As mentioned, PDL has been used in a MAS context when it was used for the B-DOING architecture [Dignum et al., 2002].

Another problem is that of *deadlines*. In SDL, deadlines are not incorporated, and Op is usually taken to mean "ought to do p now", but in reality, when one is obliged to do something, a deadline is often involved. When an agent enacting the seller role has sold an item, he is obliged to deliver the item to the buyer, but it would be inappropriate to demand it to be done immediately. A deadline is specified to indicate that the obligation should at least be fulfilled at that point in time, and otherwise, sanctions will be imposed. Dignum et al. [2003] define obligations with deadline using Logic for Contract Representation (LCR), which extends the branching-time temporal logic CTL*. An obligation in LCR to achieve p before some deadline can roughly speaking be understood as

[1] This comes from axiom F, i.e., $F(murder) \leftrightarrow O(\neg murder)$

follows. Either the agent sees to it that p happens and the obligation is fulfilled (and thus cannot be violated), or at some point the deadline occurs (without p being achieved), and the obligation has been violated. Others have proposed similar logical frameworks that attempt to incorporate deadlines in deontic logic [Aldewereld, 2009; Broersen et al., 2004; Dignum, 2004], often by using some variant of temporal logic, which naturally appeals to obligations with deadlines because of their temporal requirements. Dignum [2004] furthermore proposes conditional obligations with a deadline using LCR, making it possible to incorporate contrary-to-duty obligations (or sanctions) with deadlines.

Finally, we turn to the idea of using norms to regulate agent societies. Here, such norms are referred to as regulative norms, since they define the appropriate and expected behavior of the agents in the society. The fact is, however, that agents may join the society from different places and may be designed differently. In order to function properly in the society, a mechanism is required to ensure that the agents understand the concepts of the society in the intended way, if we expect them to behave according to the regulations. Constitutive norms are proposed as a way to define how something *counts as* something else in a society [Aldewereld, 2009; Boella and Torre, 2004; Grossi et al., 2006; Jones and Sergot, 1996; Searle, 1995]. Constitutive norms are supported by the distinction between *brute* and *institutional* facts [Searle, 1995]: Brute facts are observable things in the environment, while institutional facts are things that only exists in a society. A constitutive norm is usually written X *counts-as* Y *in* C, where X is a brute fact, Y an institutional fact, and C is a context or society. For example, the obligation to deliver an item (an institutional fact) can be completed in the context of the auction house by e.g. mailing the item or delivering it in person (brute facts). The constitutive norm thus enables agents to relate their capabilities with organizational objectives without them being identical.

We have now briefly touched upon deontic logic and its possibilities to represent normative concepts. One question that still remains unanswered is whether deontic logics as presented above really is suitable for representing norms in multi-agent systems. For example, while the paradoxes are problematic from a philosophical point of view, the same is not as convincing when pursuing practical purposes, such as ensuring that an autonomous vehicle obeys the norm to stop at red traffic lights. From a theoretical perspective, a formal logical system representing norms will be useful for a precise specification of norms, and their life cycle, to understand the impact they have on the agents. From a programming perspective, however, it seems more suitable to employ a simpler way to model norms than full-blown deontic logic, since otherwise the programmer is required to understand advanced mathematics to specify simple norms. Thus, when we investigate how to represent norms in AORTA in Chapters 4 and 5, we built upon previous work on formal specification of norms, but we keep in mind that the practical usefulness is paramount for the purpose of the AORTA framework.

3.3.2 Norms in multi-agent systems

The logical frameworks mentioned above are generally not well suited for practical purposes. We conclude this section with a discussion of more practical notion of norms and

a brief overview of how operationalization of norms has been done by other researchers.

A common approach is to define obligations as a positive constraint on an agent, prohibitions as negative constraints on an agent, and permissions as special cases that can override obligations [Criado et al., 2010; Meneguzzi and Luck, 2009]. While this is in line with our understanding of norms in general, it seems a bit too simple. It makes it possible to specify general constraints on the agent – "drive on the left side" – "do not jaywalk" – but it puts no constraints on the norms. Criado et al. [2010] distinguishes between *abstract norms* and *norm instances*, where an abstract norm can be thought of as a description of expectations, associated with an activation condition – "you are obliged to drive on the left side when you enter the UK". When the condition is met, the abstract norm is instantiated and imposed upon the agent. Others take similar approaches that make use of conditional norms [Alechina et al., 2012; Dignum, 2004; Dybalova et al., 2014; Meneguzzi and Luck, 2009; Tinnemeier, 2011]. Conditional obligations allow for a more fine-grained definition of a society, since different obligations can be applied to different agents in different contexts, so it is not surprising that much focus is put on this.

Even with conditions, obligations are still subject to the problem of *lifespan*. This is closely related to sanctioning mechanisms (at least for obligations), because *when* can the society rightfully sanction an agent for not fulfilling an obligation to do p? As mentioned in the previous section, obligations with deadlines have been investigated in the setting of deontic logic, and naturally, the same applies to practical systems. The approaches mentioned above all incorporate deadlines in their definition of norms, where a deadline is specified as partial state descriptions that define a point in time when the obligation must be fulfilled in order to avoid sanctions. Thus, a common way to define norms in practice is a tuple

$$(Deon, c, p, \delta),$$

where *Deon* is the deontic type of norm, i.e., obligation, prohibition or permission, c is the activation condition, p the content of the norm (the state of affair) and δ the deadline for fulfillment of the norm. In case of permissions and prohibitions, the deadline will usually denote the expiration time, i.e., a point in time after which the norm no longer applies. Furthermore, sanctions in case of a permission does not really make sense, as one should not be sanctioned for doing (or not doing) something one is permitted to do.

Alechina et al. [2012] note that most work on developing organization-aware agents is based on frameworks that inter-operate with existing BDI-based APLs. They argue that since such languages lack support for deliberating about norms, sanctions and deadlines, using them for programming organization-aware agents remains difficult. They propose a BDI-based APL for norm-aware agents called N-2APL (based on 2APL) with support for norms, sanctions and deadlines. N-2APL agents are norm-aware rational, which means they commit to an optimal set of plans that takes personal goals, norms and sanctions into consideration. Norm-aware agents in N-2APL deliberate using a scheduler, which take execution time and deadlines into account to order plans and action execution in an optimal order. The use of execution time enables the agents to better reason about their goals in a manner that ensures most of its goals are achieved and norms are fulfilled, but in cases where the execution time is not known a priori, the approach falls short.

Dybalova et al. [2014] have integrated N-2APL with 2OPL [Dastani et al., 2009], an organization-oriented language that defines a normative organization using brute facts, effects, counts-as rules and sanction rules. The environment is represented by the brute facts; effects determine what happens when an action is executed (similar to STRIPS actions [Fikes and Nilsson, 1971]); counts-as rules are the constitutive norms; and sanction rules determine the punishment imposed on agents violating the norms. The 2OPL normative organization is implemented using a tuple space, where facts and norms are represented as tuples. N-2APL agents connect to the tuple space using a custom environment implementation. Execution of actions by the N-2APL agents then use the tuple space in which violations can be detected. Since the N-2APL scheduler is based on the agent's specification, it has to match the specification of the organization closely, for the scheduler to be effective.

The MOISE$^+$ organizational model takes a different approach to normative constraints. Norms are *soft constraints* on the agents, and are represented by the deontic dimension, which states what is permitted and obligated in the organization. The norms connect agents enacting roles with missions, stating that the agents (playing a role) are allowed to or obliged to commit to a certain mission. For example, a seller would be permitted to commit to a mission to sell an item, but obliged to commit to a mission to deliver the item, once it has been sold. Soft constraints, as opposed to hard constraints, can be violated, i.e., the agents can choose not to commit to obligatory missions. However, since norms are essentially worthless if not enforced [Grossi et al., 2007], some mechanism is required to ensure a reaction from the society when a norm is violated. MOISE$^+$ is used in combination with artifacts by Hübner et al. [2010], and they note that while detection can be automatic, evaluation and judgment requires deliberation and reasoning. In other words, detection of violations can be implemented inside the artifacts, but it is the agents' responsibility to sanction properly, once a violation has been detected.

3.4 Concluding Remarks

We investigated the current state of organizational reasoning and identified the key dimensions of organizational reasoning that the AORTA reasoning framework will focus on. We discussed different approaches to integrating organizational reasoning into APLs and concluded that for the purpose of our work, a component-based approach is most fitting, since it can be made sufficiently generic to be integrated into different platforms, but still provide agents with strong organizational reasoning capabilities. Finally, we discussed how to represent norms in an organization, first formally using deontic logic, then by discussing how it can be more practically implemented in an APL.

This concludes the first part of the thesis. In the next part, we build upon the foundations of organizational reasoning as discussed in this chapter to design an organizational reasoning component, the core of the AORTA framework.

PART TWO

ADDING ORGANIZATIONAL REASONING TO AGENTS

In this part, we present the AORTA framework. The AORTA framework consists of a component, which provides cognitive agents with organizational reasoning capabilities.

We start by introducing the AORTA metamodel, which is based on the notions of roles, objectives and obligations, and the AORTA-component, which uses the metamodel as a basis for the reasoning (Chapter 4). We show how well-known organizational models can be translated to the metamodel. We provide operational semantics for the AORTA-component, making it possible to execute agents with organizational reasoning capabilities (Chapter 5).

CHAPTER 4

Towards a Unifying Framework

In this chapter, we present the AORTA metamodel and component. The metamodel represents the organization in a way that enables agents to use the component to perform organizational reasoning. We discussed existing approaches to organizational reasoning and investigated the phases of participation in Chapter 3. Here, we use the insights gained to propose a component that is able to embrace methods proposed by other researchers in a unifying framework. In particular, the focus points of our component are as follows:

- **The agent's perspective:** By taking a component-based approach, we essentially give as much power as possible to the agent to let it decide how to use the component. This means that the autonomy of the agent is largely unaffected by the component. This might seem strange given that the job of norms is to affect the agent's behavior, but by letting the agent remain autonomous, it can decide by itself whether the norms should modify its behavior.
- **Separation of concerns:** There is a sharp distinction between the AORTA-component and the rest of the cognitive agent, providing a clear separation of concerns. Furthermore, the metamodel follows the principles of OCMAS [Ferber et al., 2003], which by themselves are based on the notion of separation of concerns.
- **Decentralization:** An efficient organization is usually one where agents in the organization cooperate. Given that our component is integrated with each individual agent, the system is inherently decentralized. It is thus paramount that our component facilitates cooperation, since it otherwise may prove to be a burden to the agent.

The chapter is based on previous work [Jensen et al., 2013a; Jensen and Dignum, 2014; Jensen et al., 2015a], and is organized as follows. We present our view of norms and how they are used in the AORTA reasoning framework in Section 4.1. In Section 4.2, we introduce the AORTA organizational metamodel, which is based on the notions of roles, objectives and norms, and is the model used by the component to reason about organizations. We present the AORTA organizational reasoning component and discuss how to use it to reason about organizations in Section 4.3.

4.1 Representing Norms

Norms in MAS are used to describe the expected behavior of the agents in the system. In the following, we focus on how to represent obligations and prohibitions in the AORTA framework (thus putting permissions in the background). In the context of intelligent

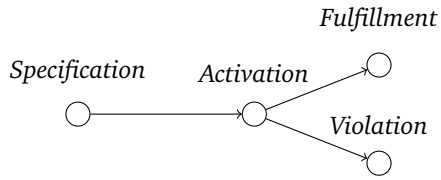


Figure 4.1: The states of an obligation. Once specified, the obligation is activated when the activation condition holds. It is fulfilled when the desirable state is achieved, and it is violated if the deadline is reached before it has been fulfilled.

agents, permissions are inherently of a different nature since they usually lift restrictions that were previously put on agents using obligations and prohibitions. For example, sanctioning in case of a permission does not really make sense, as one should not be sanctioned for doing (or not doing) something one is permitted to do.

We take the common view [Alechina et al., 2012; Dignum et al., 2002; Meneguzzi and Luck, 2009] that an obligation specifies an ideal *state* of a system, from the point of view of the creator of the obligation (the organization, in our case). An agent cannot be obliged to *do* ϕ; it will rather be obliged to *achieve* ψ (e.g., by doing ϕ, by taking some other action(s) that lead to ϕ, or by making sure ψ is achieved by some other means, such as making some another agent do it). By looking at states rather than actions, we abstract away from specific actions. This corresponds well to the fact that the same state often can be achieved in different ways, so rather than requiring agents with a specific ability to *do* something, the fulfillment of an obligation requires that the agent *achieves* something.

As discussed in Chapter 3, a common way to define norms in practice is a tuple:

$$(Deon, c, p, \delta),$$

where *Deon* is the deontic type of norm, i.e., obligation, prohibition or permission, c is the activation condition, p the content of the norm (the state of affair) and δ the deadline for fulfillment of the norm. In case of prohibitions, the deadline will usually denote the expiration time, i.e., a point in time after which the norm no longer applies. The obligation life-cycle based on such definition is shown in Figure 4.1. We distinguish between the entity associated with a conditional obligation and an activated obligation: the conditional obligation is associated with a role, denoted $O_r(p < \delta \mid c)$, and an activated obligation is associated with an agent *enacting that role*, denoted $O^a_r(p < \delta)$. For simplicity, we assume that (1) an obligation can only be activated once, (2) deadlines are persistent and (3) once fulfilled, an obligation can never be violated. In Chapter 6, when we describe the AORTA architecture, we can relax these assumptions to make the system more practically useful.

Note that even though fulfillment of an obligation is based on what the agent *achieves*, our semantics are not based on *stit*-logic [Belnap et al., 2001]. If an obligation to achieve p is only fulfilled once the agent sees to it that p holds, it may actually violate the obligation if another agent independently achieves p. For example, it would be counterintuitive to be punished for not returning a book to the library, if someone else has returned it. In essence this means that we look at obligations as parts of an organization, not as parts of the agents: an obligation is bound to a certain agent (given the role(s) it enacts), but the fulfillment of the obligation need not be done by that agent. It should, however, be fulfilled before the deadline for the obliged agent to avoid sanctions.

4.1.1 Norms in agent organizations

Even if fulfillment of a norm is not bound to the responsible agent, it seems reasonable to assume that in most cases, the responsible agent will be the one fulfilling the norm, so naturally, the agent's ability to reason about the norm is important. However, agents and their organization usually reside in an environment that none of them have full control over, so even the ability to reason about norms may not be enough if the environment is unpredictable. To accommodate this, Dignum and Dignum [2012] propose a formalism for capturing exactly the fact that agents and their organization reside in an environment that none of them fully control. Their formalism, Logic of Agent Organizations (LAO), is based on Kripke semantics to describe a system, where a world in the semantics corresponds to a state of the environment, and a transition corresponds to changes happening in the environment (e.g. agents executing actions). We briefly describe the semantics of LAO before we present our extension that allow us to use LAO to deal with obligations.

The environment and organizational concepts are described using the temporal logic CTL* to describe the environment and organizational concepts. For a set *Prop* of propositional variables, the language of LAO, $\mathscr{L}$, is based on a set $\mathscr{L}_s$ of state formulas with typical element Φ, and a set $\mathscr{L}_p$ of path formulas with typical element ϕ. We define the set of state formulas as follows:

$$\Phi ::= p \mid \neg\Phi \mid \Phi \vee \Phi \mid A\phi \mid E\phi,$$

where ϕ is a path formula and p is an atomic proposition. The set of path formulas is defined as follows:

$$\phi ::= \Phi \mid \neg\phi \mid \phi \vee \phi \mid F\phi \mid G\phi \mid X\phi \mid \phi \; U \; \phi,$$

where the operators are the usual CTL* operators: A is all, E is exists, F is future, G is always in the future, X is next, and U is until.

The semantic structure over which formulas of $\mathscr{L}$ are interpreted is a tuple

$$M = \langle W, Rt, \pi \rangle,$$

where W is a non-empty set of states, Rt is a partial order of states, and $\pi : W \to 2^{Prop}$ associates each state $w \in W$ with a set of propositional formulas true in that state.

The semantics distinguish between state and path formulas. State formulas are interpreted with regards to a state $w \in W$ and a path formula is interpreted with regards to a path through the structure given in Rt. A path in Rt is a sequence $(w_i, w_{i+1}, \dots)$, where $w_i, w_{i+1}, \dots \in W$ and $\forall i : (w_i, w_{i+1}) \in Rt$. A path is denoted $rt = (w_0, w_1, \dots)$ and we can refer to a state i in a path using $rt(i)$. For state formulas, we write $M, w \models \phi$ to denote that state formula ϕ is true in M at state w. Similarly, we write $M, rt \models \phi$ for path formulas. We let $paths(W, Rt)$ denote the set of all paths in M.

Definition 4.1 (World semantics)**.**

$$
\begin{array}{lll}
M, w \models \top & & \\
M, w \models p & \text{iff} & p \in \pi(w) \text{ where } p \in Prop \\
M, w \models \neg\phi & \text{iff} & M, w \not\models \phi \\
M, w \models \phi \vee \psi & \text{iff} & M, w \models \phi \text{ or } M, w \models \psi \\
M, w \models A\phi & \text{iff} & \forall \text{rt} \in paths(W, Rt), \text{ if } rt(0) = w \text{ then } M, rt \models \phi \\
M, w \models E\phi & \text{iff} & \exists \text{rt} \in paths(W, Rt), \text{ s.t. } rt(0) = w \text{ and } M, rt \models \phi \\
M, rt \models p & \text{iff} & M, rt(0) \models p, \quad \text{where } p \in Prop \\
M, rt \models \neg\phi & \text{iff} & M, rt \not\models \phi \\
M, rt \models \phi \vee \psi & \text{iff} & M, rt \models \phi \text{ or } M, rt \models \psi \\
M, rt \models F\phi & \text{iff} & \exists i (M, rt(i)) \models \phi \\
M, rt \models G\phi & \text{iff} & \forall i (M, rt(i)) \models \phi \\
M, rt \models X\phi & \text{iff} & M, rt(1) \models \phi \\
M, rt \models \phi \ U \ \psi & \text{iff} & \exists i \text{ s.t. } M, rt(i) \models \psi \text{ and } \forall 0 \leq k < i \ M, rt(k) \models \phi
\end{array}
$$

The semantics consider changes in the environment and the organization, but does not consider changes caused by actions performed by agents. The language, $\mathscr{L}_O$ of LAO accommodates this by adding modalities to reason about agent *capability*, *ability* (a possibility to use a capability), *attempt* (use of a capability), and *bringing about* (successful attempt). The LAO formalism can be used in connection with an organization structure to reason about the capabilities of an organization, whether it is well defined and successful, and whether it is structured such that objectives can be delegated to agents that are able to complete them. Since we look at obligations from outside the agents, the modalities are not used to define norms in our extension, thus we do not consider them here.

In the following, we present our extension of LAO: LAO with obligations (LAO_2 for short). As per Figure 4.1, norms go through various states. In the following, we provide definitions for conditional obligations with a deadline (the *specification* phase), and obligations with a deadline (the *activation* phase). We furthermore investigate how to use our definitions to represent obligations without a deadline or without a condition. Finally, we discuss some issues that arise when defining a prohibition in terms of an obligation.

In LAO_2, we extend the semantic structure with a finite, non-empty set $\mathbb{A}$ of agents and a finite, non-empty set R of roles. We furthermore add predicates for role enactment and violation to $\mathscr{L}_O$:

$$\psi ::= \phi \mid rea(a, r) \mid viol(a, r, p),$$

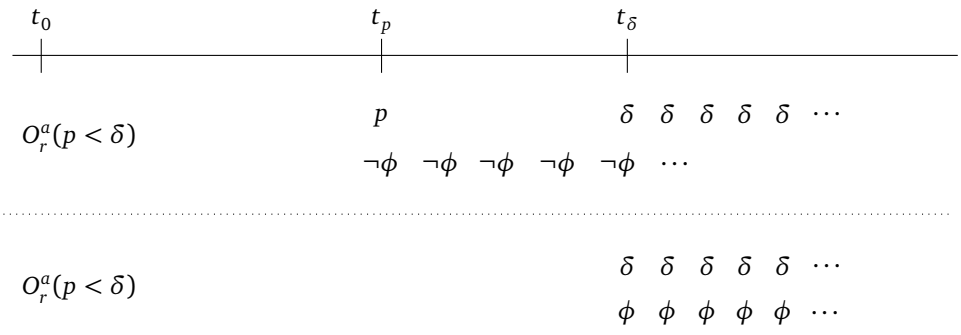


Figure 4.2: Possible sequences of obligations with a deadline in LAO_2. For simplicity, we omit the fact that p is false until t_p (and similarly for δ and ϕ).

where $rea(a, r)$ means that agent $a \in \mathbb{A}$ enacts role $r \in R$ and $viol(a, r, p)$ means that agent a has violated the obligation to achieve p while enacting role r.

We define an obligation to achieve something before a deadline using the world semantics given above. The temporal aspect of the obligation is thus captured naturally by the temporal modalities.

Definition 4.2 (Obligation with deadline)**.** An obligation for an agent $a \in \mathbb{A}$ enacting role $r \in R$ to achieve p before δ is defined as follows:

$$O_r^a(p < \delta) \equiv A((\neg p \wedge \neg\delta)\ U\ ((p \wedge AG\neg\phi) \vee (\neg p \wedge AG(\delta \wedge \phi)))),$$

where $\phi = viol(a, r, p)$.

The definition states that two different moments are relevant (as shown in Figure 4.1): *fulfillment* (when p is achieved) and *violation* (when δ is reached and p has not been achieved). In the first case, a violation can never occur, and in the second case, the violation is persistent. The possible sequences are depicted in Figure 4.2. At t_0, the obligation is activated, and both p and δ are false. At time t_p, p is achieved (in the first of the sequences) and a violation can never occur. At time t_δ, the deadline occurs, which leads to violation in the second sequence, where p was not achieved.

It is possible to define obligations without deadline as a special case of the definition above, however special care must be taken when doing so. We observe that by representing obligations with a deadline by letting $\delta = \bot$, we get:

$$O_r^a(p < \bot) \equiv A(\neg p\ U\ ((p \wedge AG\neg\phi) \vee (\neg p \wedge AG(\bot \wedge \phi)))),$$

which essentially means that a violation of the obligation cannot be detected, since the deadline never occurs. Alternatively, we let $\delta = \top$:

$$O_r^a(p < \top) \equiv A(\bot\ U\ ((p \wedge AG\neg\phi) \vee (\neg p \wedge AG(\phi)))),$$

which states that in order to fulfill the obligation, p must be the case immediately after the obligation is activated. This reduces the obligation to the usual meaning in SDL, namely that an obligation to achieve p means to achieve p now. Compared to the other special case, it is, however, better in the sense that a violation *can* be detected. In the following, we define an obligation without a deadline as follows:

$$O_r^a(p) \equiv O_r^a(p < \top).$$

Conditional obligations are inactive until the condition is reached. Furthermore, since conditional obligations are bound to roles, and obligations are bound to agents, a second condition must be met, namely that an agent is enacting the role associated with the conditional obligation.

Definition 4.3 (Conditional obligation with deadline)**.** Given an agent $a \in \mathbb{A}$, an obligation for role $r \in R$ to achieve p before δ given the condition c is defined as follows:

$$O_r(p < \delta \mid c) \equiv A(((c \wedge rea(a, r)) \rightarrow O_r^a(p < \delta))\ U\ (p \vee \delta)).$$

The definition follows the ideas of conditional obligations in LCR [Dignum, 2004], namely that c activates the obligation (for the role-enacting agent), but only until the obligation has been fulfilled (or the deadline was reached). Otherwise, the obligation could be activated again immediately after completion, if c was still the case. Note that $O_r(p < \delta \mid \top)$ is not equivalent to $O_r^a(p < \delta)$, but represents a conditional obligation that is activated as soon as an agent enacts the role.

Remark (*Uncontrollable propositions*). Our definition of obligations are not based on agency, as is often the case (e.g., in LCR [Dignum, 2004]) since obligations are defined to be tightly connected to the obligated agent. This has the (unintended) impact that we can specify norms for uncontrollable propositions (i.e., propositions that are controlled by, e.g., the environment and that happen no matter which actions the agent perform, e.g., the weather) [Boutilier, 1994]. That is, an obligation such as

$$O_r^a(rain < snow)$$

is perfectly valid and can result in either fulfillment (since it does not depend on the agent performing the action leading to the rain) or violation (since it may begin to snow before it rains). In LCR, such obligation could never be fulfilled, because it requires the agent to see to it that it rains. Since *rain* is an uncontrollable proposition, this can never happen.

We argue, however, that there is no reason (from the designer's perspective) to apply such norms to the system since the point of norms and sanctions is to modify the agent's behavior to adhere to the norms. Since the agent cannot control the weather, the fulfillment of the obligation is out of the agent's hand and there are no obvious sanctions to impose.

4.1.2 Prohibitions

The concept of prohibition (F) is often defined in terms of obligation:

$$F(p) \equiv O(\neg p).$$

It is therefore natural to investigate how well this works with our approach. By using the definition above, a prohibition is defined as follows:

$$\begin{aligned} F_r^a(p < \delta) &\equiv O_r^a(\neg p < \delta) \\ &\equiv A((p \wedge \neg\delta)\ U\ ((\neg p \wedge AG\neg\phi) \vee (p \wedge AG(\delta \wedge \phi)))), \end{aligned}$$

where $\phi = viol(a, r, \neg p)$. There are two important issues with this representation of prohibitions. First, the prohibited state of affair p should initially be the case, and second, if p becomes false at some point in time before the deadline occurs, the prohibition can never be violated. We believe a more natural interpretation of a prohibition is one that disallows p *until* the prohibition expires, i.e., when δ is the case. We thus propose the following definition of a prohibition with expiration:

Definition 4.4 (Prohibition with expiration)**.** Given a role r and an agent $a \in \mathbb{A}$ enacting $r \in R$, a prohibition to achieve p until δ, denoted $F_r^a(p < \delta)$, is defined as:

$$F_r^a(p < \delta) \equiv A(\neg\delta\ U\ ((\neg p \wedge AG(\delta \wedge \neg\phi)) \vee (p \wedge (\phi\ U\ AG(\delta \wedge \phi))))),$$

where $\phi = viol(a, r, p)$.

The definition makes two points in time explicit: when the prohibition expires without being violated (and can never be violated), and when the prohibition is violated. In that case, we the prohibitions it is violated until the expiration (and will then forever be violated), since we require that the prohibition is in effect even if it has been violated (e.g., running a red light does remove the prohibition of running a red light). The special case where $\delta = \top$ results in a prohibition that is either immediately fulfilled or violated, and when $\delta = \bot$, the prohibited state is always forbidden (since the deadline never occurs).

Conditional prohibitions can be defined similarly to conditional obligations.

Definition 4.5 (Conditional prohibition with expiration)**.** Given an agent $a \in \mathbb{A}$, a prohibition for role $r \in R$ to refrain from achieving p until δ given the condition c is defined as follows:

$$F_r(p < \delta \mid c) \equiv A(((c \wedge rea(a, r)) \rightarrow F_r^a(p < \delta))\ U\ \delta).$$

Note that the last sub-expression of conditional prohibition differs from that of the conditional obligation, since the prohibition is active until it expires (even if it has been violated).

4.1.3 Handling contrary-to-duty paradoxes

We have previously argued that sanctions can be modeled as contrary-to-duty obligations. For example, if the obligation to be verified before participating in an auction is violated, the agent is obliged to leave the auction:

$$O_{customer}(left\_auction < \top \mid viol(a, customer, verified)).$$

Note that given the deadline, the obligation should be fulfilled immediately, corresponding well to the intuitive understanding of such sanction. However, as we saw in Chapter 3, contrary-to-duty obligations may lead to paradoxes in SDL. We briefly show how this is not an issue in LAO_2. There are many different contrary-to-duty situations that could be investigated, but our intention is not to investigate all of them (e.g., Dignum [2004] investigates three versions of Chisholm's paradox for the LCR formalism). In the following, we show how LAO_2 handles Forrester's paradox.

Example 4.6 (Forrester's paradox in LAO_2)**.** *Recall that Forrester's paradox states that it is forbidden to commit murder, but if you do commit a murder, you ought to do it gently. Given an agent $a \in \mathbb{A}$ and a role $r \in R$, we have the following specification:*

1. $F_r^a(murder)$
2. $O_r(murdergently \mid murder)$
3. $murdergently \rightarrow murder$
4. $murder$

In SDL, this leads to the conclusion that a is obliged to murder. In LAO_2, violations are explicitly represented and thus we do not reach a contradictory state. Let state S_1 represent a state where the norms hold but no actions have been performed. In the next state, the action is performed and one of two things happens: either the agent performs the murder gently, or he does not. Figure 4.3 shows the possible state transitions. In state S_{2a}, the agent violates the norm that prohibits murder, but fulfills the norm of murdering gently. In state S_{2b}, the agent violates both norms. However, in none of the states is it possible to derive $O_r^a(murder)$.

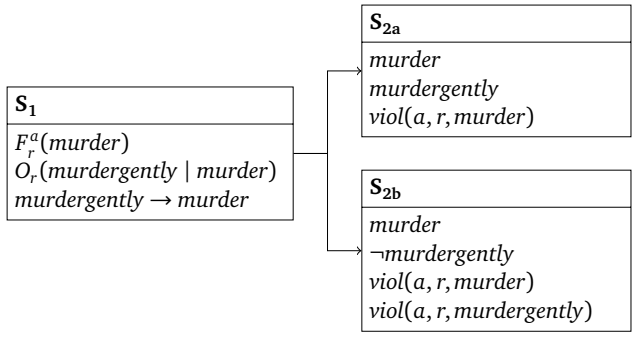


Figure 4.3: *Forrester's paradox in LAO_2.*

4.2 AORTA Organizational Metamodel

We base organizational reasoning in AORTA on an organizational *metamodel*. A metamodel is a model of a model, which in this case makes the AORTA metamodel a model of organizational models. That is, our metamodel highlights properties of existing organizational models in a way that makes it possible for agents to use the metamodel for reasoning about existing organizational models. This is in line with our component-based approach that provides cognitive agents with organizational reasoning capabilities: the component allows different kinds of cognitive agents to perform organizational reasoning, and the metamodel allow the reasoning to happen using different kinds of organizational models.

While it would be possible to add support for multiple organizational models rather than creating a metamodel, we claim that by taking this approach, we make the reasoning process as simple as possible. Furthermore, from a practical point of view, a common metamodel makes it possible to work with different organizations without having to learn new languages or concepts.

In Chapter 2, we noted that especially concepts such as roles, objectives and norms are key points in existing organizational models. Furthermore, to relate agents in the organization, it is often possible to partition agents into groups or to define hierarchical relations between the agents. The AORTA organizational metamodel is thus based on roles, objectives, norms and hierarchical relationship between roles.

Definition 4.7 (Metamodel)**.** An AORTA organizational metamodel consists of a number of predicates describing the core concepts of an organizational model.

role(R, Os)	R is the role name, and Os is a set of objectives, where each O is a partial state description.
obj(O, S)	O is the name of an objective, and S is a set of sub-objectives.
dep(R_1, R_2, O)	Role R_1 depends on role R_2 for completion of objective O.
rea(A, R)	Agent A enacts role R.
cond($R, Deon, O, D, C$)	A conditional norm for role R that activates when C holds. *Deon* is *obliged* or *forbidden*. The state of affair is O, and the expiration or deadline is D.
norm($A, R, Deon, O, D$)	An norm for agent A playing role R. *Deon* is *obliged* or *forbidden*. The state of affair is O, and the expiration or deadline is D.
viol($A, R, Deon, O$)	Agent A playing role R has violated the norm of type *Deon* concerning O.

A role is defined only by its name and its *main* objectives. Sub-objectives of an objective are specified using obj-predicates. We distinguish between the different states of obligations by using different predicates. For example, consider the following conditional obligation:

$$O_{buyer}(verified(Me) < bid(Me, Item) \mid registered(Me)),$$

where we assume *Me* is a variable that binds to the agent's name. In the metamodel, it is represented by the predicate

$$\mathsf{cond}(buyer, obliged, verified(Me), \\ bid(Me, Item), registered(Me)).$$

If an agent *bob* enacts the *buyer* role and registers in the auction house, the obligation is activated and we have

$$O^{bob}_{buyer}(verified(bob) < bid(bob, Item)).$$

This is represented by the predicate

$$\mathsf{norm}(bob, buyer, obliged, verified(bob), bid(bob, Item)).$$

A violation of the norm can then be represented by the following predicate:

$$\mathsf{viol}(bob, buyer, obliged, verified(bob)).$$

While the metamodel can be used as-is, we further argue its usefulness by the fact that existing organizational models can be translated into it. As a proof of concept, we show that parts of the OperA model and the MOISE$^+$ model can be translated to the metamodel in Appendix B.

We argue that our metamodel conforms with the OCMAS principles proposed by Ferber et al. [2003]:

1. **Separation of "what" and "how":** The objectives define the organizational purpose, i.e., what agents entering the organization are expected to achieve. Though norms may preclude certain ways of achieving an objective, the metamodel does not specify precisely how objectives should be achieved.

2. **No agent description:** Agents are specifically mentioned in the rea and norm predicates. However, no assumptions about the agents are made. Role enactment relates an agent to a role, and norms provide expectations to the agent.

3. **Context of interaction:** Though the metamodel makes no explicit use of groups, the use of dependencies allows us to specify hierarchical relationships between roles. This makes it possible to define how agents interact in order to complete objectives (e.g., by using delegation).

Furthermore, the metamodel express a normative view of the system by the use of conditional norms. As mentioned, the point of such norms is to affect the agent's behavior such that it lies within the expectations of the organization.

Remark (*Using norms to represent a timeline*). We claim that the use of norms can be used to express a different set of expectations than the normative ones. For example, in our translation from the OperA model to the metamodel, we translate scenes and landmarks into conditional norms. Interaction in OperA is described by an ordering of scenes. For example, scene 1 should happen scene 2 and so on. Furthermore, each scene consists of a number of landmarks that should happen in a specific order. For example, landmark λ_1 should be completed before λ_2 and λ_3, and so on.

We can translate this into norms, where, e.g., λ_1 is the objective and the conjunction, $\lambda_2 \wedge \lambda_3$ is the deadline. By relating this to the roles responsible for each objective, we can create a set of conditional norms that describe the expected order of completion.

We provide full details of the translation in Appendix B, where we furthermore show how to translate MOISE$^+$ missions into conditional norms.

We round this section off with an example of a metamodel. In the next section, we present the AORTA-component, which enables agents to operationalize the metamodel.

Example 4.8 (Metamodel of the Electronic Auction House)**.** *We showed in Chapter 2 how the EAH scenario could be modeled using the OperA organizational model. Here, we show how the part of the scenario concerning registration and verification can be modeled using the AORTA organizational metamodel. We refer to Appendix A for the complete implementation of the scenario in the metamodel (using the metamodel language introduced in Chapter 6).*

The metamodel is shown in Table 4.5. The roles are defined by their name and main objectives. Each objective is defined by its sub-objectives. For example, a sub-objective of becoming a verified customer is becoming a registered customer. Furthermore, the customer depends on the manager for the verification process, and the customer is obliged to become registered before he can be verified. The manager should refrain from verifying agents that are known to provide bad credentials.

Table 4.5: *Metamodel for a part of the EAH scenario.*

role(*customer*, [*registered*(*Agent*), ...])
role(*manager*, [*verified*(*Agent*)])

obj(*registered*(*Agent*), [])
obj(*verified*(*Agent*), [*registered*(*Agent*)])

dep(*customer*, *manager*, *verified*(*Agent*))

cond(*customer*, *obliged*, *registered*(*Me*), *verified*(*Me*), *agent*(*Me*))
cond(*manager*, *forbidden*, *verified*(*Ag*), $\bot$, *badInfo*(*Ag*) $\wedge$ *registered*(*Ag*))

4.3 The AORTA Component

The core part of AORTA is a component that, when added to an agent, provides it with organizational reasoning capabilities. It assumes a preexisting organization (specified using the metamodel), is independent from the agent, and focuses on reasoning rules that specify how the agent reasons about the organization. The organization is completely separated from the agent, meaning that the architecture of the agent is independent from the organizational model, and the agent is free to decide on how to use AORTA in its reasoning. The separation is possible because reasoning is based on the organizational metamodel and because the component is designed to be non-intrusive (i.e., it does not change the agent's existing reasoning mechanisms).

Organizational reasoning in AORTA divided into three phases: *norm check* (NC), *option generation* (OG) and *action execution* (AE). The NC-phase uses the agent's mental state and organizational state to determine if norms are activated, satisfied, expired or violated, and updates the organizational state accordingly. The OG-phase uses the organizational specification to generate possible organizational options. The agent considers these options in the AE-phase using reasoning rules, which can alter the organizational state, the agent's intentions or send messages to other agents. The component is shown in Figure 4.6. We assume it is connected to a *cognitive* agent, i.e., an agent with mental attitudes (such as beliefs and goals) and practical reasoning rules. This could, for example, be a BDI agent (defined by its mental attitudes: beliefs, desires and intentions), which has been used as a basis for many agent programming languages [Bordini et al., 2007; Dastani, 2008; Hindriks, 2009].

In the following subsections, we provide an overview of each of the phases. We do not go into too much detail concerning the execution of each phase here, but rather present their general idea and purpose. In the next chapter, we present the operational semantics

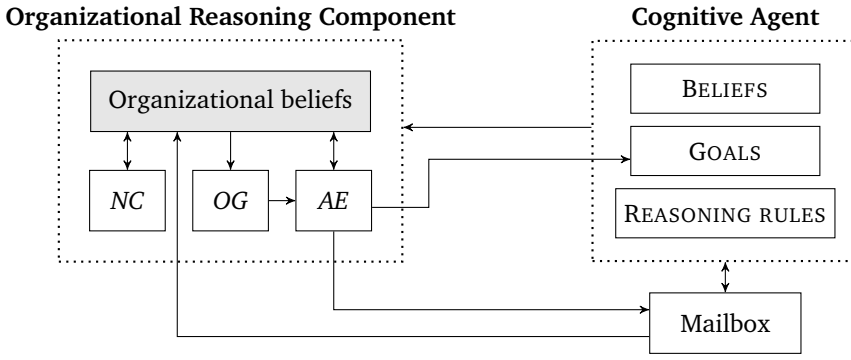


Figure 4.6: The AORTA component. The arrows indicate flow of information. Norms are updated and options are generated based on the organizational beliefs, and actions are based on the generated options.

of the component, which will precisely define the functioning of each phase.

4.3.1 Norm check

The NC-phase is the first phase of the organizational reasoning component. The component uses the agent's state to determine for each norm if it should change to a new state (cf. the states in Figure 4.1 and the semantics of obligations and prohibitions). That is, if the condition for activating a norm has occurred, the component activates the norm by updating the organizational state. Similarly, it checks whether obligations are fulfilled (the objective is completed) or violated (the deadline was reached before the objective was completed), and whether prohibitions have expired (the prohibited state was not reached and the expiration time has occurred) or have been violated (the prohibited state was reached before the prohibition expired).

Example 4.9 (Norm check)**.** *An agent,* bob*, enters the EAH to bid on an item. Initially, the following conditional norms apply:*

$$\begin{aligned}&\mathsf{cond}(\mathit{customer}, \mathit{obliged}, \mathit{registered}(\mathit{Me}), \mathit{verified}(\mathit{Me}), \mathit{agent}(\mathit{Me}))\\&\mathsf{cond}(\mathit{borrower}, \mathit{obliged}, \mathit{verified}(\mathit{Me}),\\&\qquad \mathit{bid}(\mathit{Me}, \mathit{Item}), \mathit{registered}(\mathit{Me})).\end{aligned}$$

Next, the agent performs the sequence shown in Figure 4.7[1]*. First, he enacts the* customer *role, which activates the first norm. He adheres to the norm by registering, and then enacts the* buyer *role. This activates the second norm, which requires that the agent has been verified before bidding on items. The agent then bids on an item in an auction, which violates the norm.*

$\rightarrow$ enact(*customer*)
$\leftarrow$ norm(*bob*, *buyer*, *obliged*, *registered*(*bob*), *verified*(*bob*))
$\rightarrow$ register(*bob*, *Details*)
$\rightarrow$ enact(*buyer*)
$\leftarrow$ norm(*bob*, *buyer*, *obliged*, *verified*(*bob*), *bid*(*bob*, *Item*))
$\rightarrow$ bid(*bob*, *"PH-lamp"*)
$\leftarrow$ viol(*bob*, *buyer*, *obliged*, *verified*(*bob*))

Figure 4.7: *An example of the NC-phase. Arrows pointing right (→) indicate an action executed by the agent. Arrows pointing left (←) indicate the NC-phase updating the norms of the system.*

[1] Note that several steps of reasoning have been omitted, since the example focuses on the NC-phase. For example, we do not consider how the agent decides to register in the first place

4.3.2 Option generation

In the OG-phase, the component uses the mental state of the agent and the organizational state to consider what the agent can do regarding the organization. The following organizational aspects are considered in the OG-phase:

Role enactment: Roles that are possible to enact given the agent's capabilities and goals. An agent has a capability ϕ if the agent has one or more plans to achieve ϕ.

Role deactment: (Currently enacting) roles that have been fulfilled (e.g., all objectives have been completed) or are no longer useful (e.g., the norms limits the agent).

Norms: Active norms bound to the agent.

Violations: Norms that have been violated.

Delegation: Objectives that can be delegated based on a dependency relation (e.g., delegating the verification process to an agent enacting the manager role).

Information: Obtained information that other agents will benefit from knowing (e.g., informing the customer about the result of the verification process).

Once the options are generated, they are made available to act upon in the final phase, action execution.

Example 4.10 (Option generation)**.** *We consider the same scenario as in Example 4.9, but this time with a focus on option generation. The agent,* bob*, enters the EAH. Based on the plans he has available, he has the capability to register (i.e., one of his plans achieves registered*(*bob*)*) and to bid on items. Based on the capabilities and on the objectives of each role, options 1 and 2 of Table 4.8 are generated where each is an option to enact a role, specified in the* option language *(see next chapter).*

We saw in the previous example that he enacts the customer *role (thus he acts upon option 1. Since the NC-phase activates the norm to register, option 3 is generated. As we know, he chooses to first register (option 3) and then enact the* buyer *role (option 2). Then, he violates the activated norm (option 4) to verify his credentials before bidding, which generates the final option.*

In this example, the agent does not act on the violation of the norm.

Table 4.8: *An example of the OG-phase. Each item is an option that was generated based on the state of the agent and the organization.*

(1) role(*customer*)
(2) role(*buyer*)
(3) norm(*customer*, *obliged*, *registered*(*bob*), *verified*(*bob*)).
(4) norm(*buyer*, *obliged*, *verified*(*bob*), *bid*(*Auction*, *Bid*, *bob*)).
(5) viol(*bob*, *customer*, *obliged*, *verified*(*bob*)).

4.3.3 Action execution

The AE-phase uses reasoning rules to decide how to react on an option in a given context. The AE-phase selects at most one option to act upon. The reasoning is based on rules of the form[2]

$$option \ : \ context \rightarrow action$$

where *option* is a previously generated option, *context* is a state description that should hold for an action to be applicable, and *action* is the action to be executed.

The agent has actions available to enact or deact a role, commit to complete or drop an objective, and send messages. This corresponds well to the options that can be generated in the previous phase.

Example 4.11 (Action execution)**.** *We again consider the scenario described in Example 4.9. The agent uses the action rules listed in Table 4.9 to decide how to act upon the options generated in the previous phase:*

(1) When an option to enact the customer role is generated, the agent simply enacts the role (the use of $\top$ *in the context means that the rule is always applicable). Since agents in the example are tailored to the EAH, it seems reasonable to allow agents to enact the customer role if they are capable of it.*

(2) The option to enact the buyer role is acted upon if the agent has already registered. This is a matter of choice, and it would also be possible to consider a similar rule, applicable only if the agent has a goal to buy an item.

(3) The agent chooses to adhere to the norm to register if he has a goal to buy or sell an item.

(4) The final rule does not consider any options, and is thus always applicable if the context matches the current state. It states that if the agent desires to buy an item and believes it has provided bad registration info, it bids without waiting for verification.

As we know, the agent (intentionally) violates the verification norm and has no rules to act upon the violation option.

Table 4.9: *An example of action rules in the AE-phase.*

(1) $\mathsf{role}(customer) \ : \ \top \rightarrow \mathsf{enact}(customer)$

(2) $\mathsf{role}(buyer) \ : \ \mathbf{bel}(registered(bob)) \rightarrow \mathsf{enact}(buyer)$

(3) $\mathsf{norm}(customer, obliged, registered(bob), verified(bob)) \ :$
$\mathbf{goal}(bought(Item)) \vee \mathbf{goal}(sold(Item)) \rightarrow \mathsf{commit}(registered(bob))$

(4) $\top \ : \ \mathbf{goal}(bought(Item)) \wedge \mathbf{bel}(badInfo(bob)) \rightarrow \mathsf{commit}(bid(Item))$

[2]Inspired by the plan syntax of AgentSpeak(L) [Rao, 1996].

4.4 Concluding Remarks

We have taken the first steps towards a unifying framework for organization-aware agents. We presented our view of norms in agent organizations and provided formal semantics for conditional obligations and prohibitions. We furthermore proposed the AORTA metamodel, which is designed to incorporate the most commonly used concepts of organizational modeling to allow agents to reason about different organizational models. Finally, we presented the AORTA-component, which, when added to cognitive agents, allow them to reason about the organizational metamodel.

CHAPTER 5

Operational Semantics

In Chapter 4 we presented the AORTA *reasoning component*, which handles norms, generates options and executes actions. The focus of this chapter is to formally describe how each of these phases work. We do so by defining the *structural operational semantics* of agents with an AORTA-component (previously described in [Jensen and Dignum, 2014; Jensen et al., 2015a]). We define the semantics using a transition system [Plotkin, 1981], which consists of a set of transition rules defining the transformation from one configuration to another.

The purpose of operational semantics is to describe how a computation is performed. Depending on how fine-grained the semantics are specified, this can then be used when implementing the system. In a programming language, this could, e.g., be used for an assignment statement $v := e$, to describe the steps it performs: first evaluate the expression e and then change the value bound to the variable v to the result of the evaluation. In our case, the operational semantics are used to describe the steps performed by the AORTA-component to go through the reasoning phases: norm check, option generation and action execution[1]. We show how they can be used to execute AORTA-agents in this chapter, and in Chapter 6, we describe how to implement the AORTA architecture based on the operational semantics.

In the previous chapter, we formalized the intended behavior of organization-aware agents using temporal logic and in this chapter, our goal is to capture this behavior using operational semantics. Note that there is currently no formal connection between these semantics, and future work is needed in order to establish the correctness of the operational semantics with respect to the semantics of obligations.

The chapter is organized as follows. We begin by introducing the concepts of knowledge bases and mental state in Section 5.1. We present the AORTA-agent configuration and its semantics (based on the norm semantics introduced in the previous chapter) in Section 5.2. The transition system for the AORTA-component is presented in Section 5.3 and finally, we show how to execute AORTA-agents in Section 5.4 using the EAH scenario.

5.1 Mental State

Reasoning in AORTA is based on reasoning rules that can be applied in certain states to execute organizational actions. Each state is represented by the facts believed to be true in that state, so, e.g. in one state, an auction for a lamp is created, and in another, the auction has finished. The reasoning rules thus describe the state(s) in which a given rule

[1] This is similar to work done by [Tinnemeier, 2011].

is applicable, based on the facts of those states. These facts are grouped into knowledge bases with facts of the same type. Here, a type could be each of the mental attitudes depicted in Figure 4.6, i.e., the goals and beliefs, and the organizational beliefs in the reasoning component. A knowledge base thus holds a number of facts that the agent can use in its reasoning process. To make the integration with cognitive agent systems as painless as possible, we describe the facts in the knowledge bases using a predicate language, L, following the ideas of existing agent systems [Bordini et al., 2007; Dastani, 2008; Hindriks, 2009]. We furthermore define sub-languages, L^{org} for the organizational metamodel and L^{opt} for organizational options.

One of the key ideas of AORTA is the notion of the organizational knowledge base used by the component for reasoning about options and actions, and an *options base* containing the options generated in the OG-phase.

Definition 5.1 (Organizational language)**.** The organizational language, denoted $L^{org} \subset L$ with typical element γ, is based on the organizational metamodel. It is defined by the following predicates:

$$\begin{aligned}\gamma \quad ::= \quad & \mathsf{role}(R, Os) \mid \mathsf{obj}(O, Os) \mid \mathsf{dep}(R_1, R_2, O) \mid \mathsf{rea}(A, R) \mid \\ & \mathsf{cond}(R, Deon, O, D, C) \mid \mathsf{norm}(A, R, Deon, O, D) \mid \mathsf{viol}(A, R, Deon, O).\end{aligned}$$

The terms of the predicate are variables (indicated by beginning with an uppercase letter) of L.

Definition 5.2 (Options language)**.** The option language, $L^{opt} \subset L$, with typical element γ, is defined as follows:

$$\begin{aligned}\gamma \quad ::= \quad & \top \mid \mathsf{role}(R) \mid \neg\mathsf{role}(R) \mid \mathsf{obj}(R) \mid \neg\mathsf{obj}(R) \mid \mathsf{norm}(R, Deon, O, D) \mid \\ & \mathsf{viol}(A, R, Deon, O) \mid \mathsf{send}(R, ilf, Msg),\end{aligned}$$

with variables of L and $ilf \in \{tell, achieve\}$.

Having constrained the organization and option language, we can define the knowledge bases used by AORTA. As mentioned, we distinguish between organizational beliefs and options, agent beliefs and goals.

Definition 5.3 (Knowledge bases)**.** We build each knowledge base using the predicate language, L. The agent's belief base and goal base are denoted Σ_a and Γ_a, respectively. The organizational specification and options are denoted Σ_o and Γ_o, respectively. We have that $\Sigma_a, \Gamma_a \subseteq L$, $\Sigma_o \subseteq L^{org}$ and $\Gamma_o \subseteq L^{opt}$. A knowledge base contains only *ground* predicates.

The mental state of an agent using the AORTA-component can then be defined in terms of the knowledge bases.

Definition 5.4 (Mental state)**.** An AORTA mental state, *MS*, is defined as a tuple of knowledge bases:

$$\text{MS} = \langle \Sigma_a, \Gamma_a, \Sigma_o, \Gamma_o \rangle.$$

We write **MS** to denote a set of mental states.

Each of the knowledge bases in the mental state can be queried using *reasoning formulas*.

Definition 5.5 (Formulas)**.** AORTA uses reasoning formulas, L_R, with typical element ρ, which are based on organizational formulas, option formulas, belief formulas and goal formulas.

$$\rho \ ::= \ \top \mid \mathbf{org}(\phi) \mid \mathbf{opt}(\phi) \mid \mathbf{bel}(\phi) \mid \mathbf{goal}(\phi) \mid \neg\rho \mid \rho_1 \wedge \rho_2,$$

where $\phi \in L$.

Organizational formulas, $\mathbf{org}(\phi)$, are used to query the organizational belief base, option formulas, $\mathbf{opt}(\phi)$, query the options base, belief formulas, $\mathbf{bel}(\phi)$, query the belief base and goal formulas, $\mathbf{goal}(\phi)$, query the goal base. For example, the formula

$$\mathbf{org}(\text{rea}(\mathit{bob}, \mathit{buyer})) \wedge \mathbf{goal}(\mathit{bought}(\text{“PH-lamp”}))$$

succeeds if Bob enacts the *buyer* role and he has *bought*(“PH-lamp”) as a goal. We can formally define what it means that a reasoning formula succeeds by defining its semantics.

Definition 5.6 (Semantics of reasoning formulas)**.** The semantics of reasoning formulas is based on the agent's mental state, $MS = \langle \Sigma_a, \Gamma_a, \Sigma_o, \Gamma_o \rangle$.

$$\begin{array}{lll}
MS \models \top & & \\
MS \models \mathbf{bel}(\phi) & \textit{iff} & \phi \in \Sigma_a \\
MS \models \mathbf{goal}(\phi) & \textit{iff} & \phi \in \Gamma_a \\
MS \models \mathbf{org}(\phi) & \textit{iff} & \phi \in \Sigma_o \\
MS \models \mathbf{opt}(\phi) & \textit{iff} & \phi \in \Gamma_o \\
MS \models \neg\rho & \textit{iff} & MS \not\models \rho \\
MS \models \rho_1 \wedge \rho_2 & \textit{iff} & MS \models \rho_1 \text{ and } MS \models \rho_2
\end{array}$$

The semantics makes it possible to reason about the agent's mental state, using formulas such as the example above.

5.2 Agent Configuration

An agent is defined by its *configuration*. The configuration contains its current mental state, its capabilities, mailbox and rules, and by changing it, the agent is changed. It can thus be seen as the state of the agent. For example, by executing an action or by receiving a message, the result is a new (state of the) agent. In most cases, the mental state will be changed resulting in a configuration (or state) change. In the following, we define each of the parts of the configuration before finally defining the agent configuration.

We define organization-specific actions, that are used to perform changes to the organization, committing to objectives and coordinate with other agents in the organization.

Definition 5.7 (Actions)**.** A set of actions with typical element a is denoted Act.

$$a ::= \mathsf{enact}(R) \mid \mathsf{deact}(R) \mid \mathsf{commit}(O) \mid \mathsf{drop}(O) \mid \mathsf{send}(A, M),$$

where R is a role, O is an organizational objective, A is an agent and M is a message (which is a reasoning formula).

Agents can decide when to execute a given action based on action rules. Action rules match options generated in the OG-phase, and are applicable if the context follows from the agent's mental state.

Definition 5.8 (Action rule)**.** A set of action rules is defined by:

$$R_A = \{o \ : \ \rho \rightarrow a \mid o \in L^{opt}, \rho \in L_R, a \in \mathsf{Act}\}$$

Consider, for example, the following rule for the agent Bob:

$$\mathsf{role}(\mathit{buyer}) \ : \ \mathbf{org}(\mathsf{rea}(\mathit{bob}, \mathit{customer})) \wedge \mathbf{goal}(\mathit{bought}(\mathit{Item})) \rightarrow \mathsf{enact}(\mathit{buyer}).$$

It is applicable when the OG-phase has generated the option to enact the *buyer* role, and the agent already enacts the *customer* role and has the goal to buy an item (i.e., $\mathsf{rea}(\mathit{bob}, \mathit{customer})$ is in the organizational belief base and $\mathit{bought}(\mathit{Item})$ is in the goal base).

Actions are executed using the *action transition function*, which defines how each action alters the mental state.

Definition 5.9 (Action transition function)**.** An action transition function, $\mathscr{A}$, is defined as $\mathscr{A} : (\mathsf{Act} \times \mathbf{MS}) \rightarrow \mathbf{MS}$.

In the following, A is the agent's name. For each action, we omit the parts of the

mental state that are not changed by the execution.

$$\mathscr{A}(\mathsf{enact}(R), MS) = \langle \Sigma_o \cup \{\mathsf{rea}(A,R)\}, (\Gamma_o \cup \{\mathsf{send}(\top, \mathit{tell}, \mathbf{org}(\mathsf{rea}(A,R)))\}) \setminus \{\mathsf{role}(R)\}\rangle \text{ if } \Sigma_o \models \mathsf{role}(R, Os) \text{ and } \Sigma_o \not\models \mathsf{rea}(A,R)$$

$$\mathscr{A}(\mathsf{deact}(R), MS) = \langle \Sigma_o \setminus \{\mathsf{rea}(A,R)\}, (\Gamma_o \cup \{\mathsf{send}(\top, \mathit{tell}, \neg\mathbf{org}(\mathsf{rea}(A,R)))\}) \setminus \{\neg\mathsf{role}(R)\}\rangle \text{ if } \Sigma_o \models \mathsf{rea}(A,R)$$

$$\mathscr{A}(\mathsf{commit}(\phi), MS) = \langle \Gamma_a \cup \{\phi\}\rangle \text{ if } \Sigma_a \not\models \phi \text{ and } \Gamma_a \not\models \phi$$

$$\mathscr{A}(\mathsf{drop}(\phi), MS) = \langle \Gamma_a \setminus \{\phi\}\rangle \text{ if } \Gamma_a \models \phi$$

Notice that the execution of an enactment action generates an option to inform others about the enactment. We use $\top$ to denote that the message does not have an intended recipient role; it is up to the agent to decide who to inform, if it deems this necessary.

Note that once the agent enacts a role, the corresponding option is removed from the options-base. Since only the AORTA-component can change the organizational beliefs, and since the agent cannot enact a role it already enacts, this seems reasonable[2]. Also note, that the same does not hold for committing to objectives, since the cognitive agent using the AORTA-component also has the possibility to make changes to the goal base.

The action transition function does not handle the send-action, as it does not make changes to the mental state. Instead, it puts messages into the agent's outbox, which then processes the message. The agent furthermore has an inbox for receiving messages from other agents. Since some agents may not be trustworthy, we provide a message transition function that can be used to filter incoming messages.

Definition 5.10 (Message transition function)**.** A message transition function, $\mathscr{M}$, is defined as follows:

$$\mathscr{M} : Ag \times \phi \times \mathbf{MS} \rightarrow \mathbf{MS},$$

where Ag is the set of agents, and ϕ is the content of the message.

We do not go into details with how to filter incoming messages, but note that, e.g., the reputation of the agent might be a reasonable indicator for whether to accept the message [Grossi et al., 2007; Hübner et al., 2009].

Finally, following Padgham and Lambrix [2005], we define the capabilities of an agent as the states the agent can achieve.

[2]Of course, it is possible to construct reasoning rules of the form $o : \rho \rightarrow \mathsf{enact}(R)$, where $o \neq \mathsf{role}(R)$, i.e., where the agent did not act upon a role option. However, since the enact-action is only applicable for roles the agent does not currently enact, we still find it reasonable to remove the option.

Definition 5.11 (Capability)**.** A set of capabilities for an agent α is defined as follows:

$$\mathsf{cap}(\alpha) = \{\phi \mid \mathit{ableToAchieve}(\alpha, \phi)\},$$

where $\mathit{ableToAchieve}(\alpha, \phi)$ means that the agent is able to achieve ϕ.

What "able to achieve" means depends on the cognitive agent. For example, in [Riemsdijk et al., 2011] it is suggested that the capability for a GOAL agent to achieve a goal ϕ translates to the existence of goal conditions in action rules and context conditions of modules. In BDI systems with a plan library that associates plans with a triggering event (e.g. *Jason*), an agent has the capability to achieve a goal ϕ if it has at least one plan with a triggering event of type *achieve goal* ϕ.

We are now able to define the agent configuration:

Definition 5.12 (Agent configuration)**.** An AORTA-agent configuration is defined by the following tuple:

$$A = \langle \alpha, \mathrm{MS}, AR, \mathscr{A}, \mathscr{M}, C, \mu \rangle,$$

where α is the name of the agent, *MS* is a mental state, *AR* is the agent's reasoning rules, $AR \subseteq R_A$, $\mathscr{A}$ and $\mathscr{M}$ are action and message transition functions, respectively, $C = \mathsf{cap}(\alpha)$ is the agent's set of capabilities, and $\mu = \langle \mu_{in}, \mu_{out} \rangle$ is the agent's mailbox, containing incoming and outgoing messages.

In the previous chapter, we proposed LAO_2 as a way to formally express norms in agent organizations. We used a temporal logic to capture the fact that the state of a norm change over time. In this chapter, we propose a small change to the world semantics of LAO_2, such that propositions in a state is true *iff* it is true in the agent's mental state. In other words, we use LAO_2 to express the agent's state over time, allowing us to express 1) norms about the agent's mental attitudes and 2) changes to the agent state. Specifically, we make three changes. First, we let π denote a function that associates each state with the agent's mental state, $\pi : W \rightarrow \mathbf{MS}$, second, we change the semantics structure to include the agent configuration:

$$M = \langle W, Rt, \pi, Ag, R, A \rangle,$$

and third, we change the semantics of atomic propositions in LAO_2:

$$\begin{array}{lll} M, w \models p & \text{iff} & \pi(w) \models p, \text{ where } p \in L_R \\ M, rt \models p & \text{iff} & M, rt(0) \models p, \text{ where } p \in L_R \end{array}$$

The result is a formal model that allows us to reason about the agent's beliefs about the world over time and how the norms of the organization affect those beliefs. Recall that the semantic structure includes a set of agents and a set of roles. Together with the agent configuration, this allows us (for each agent) to detect changes in the norm life cycle of all the agents in the system, making it possible for the agents to, e.g., sanction each other,

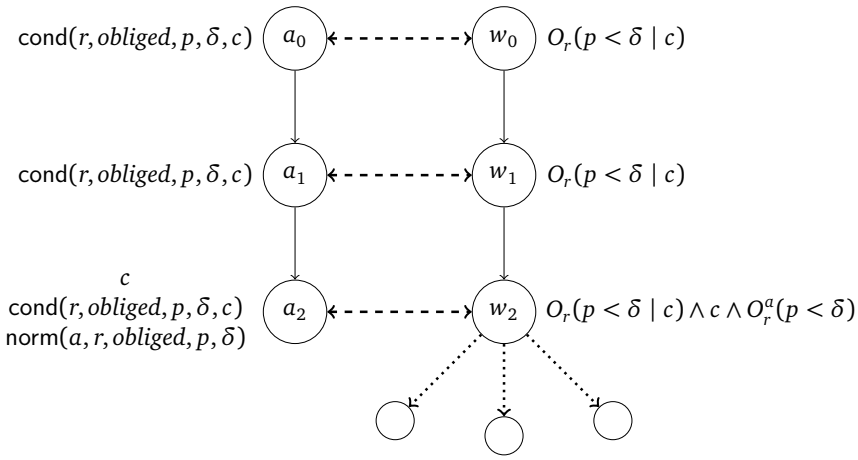


Figure 5.1: The correspondence between the agent's mental state and the world semantics. A dashed line between states indicate correspondence, and dotted lines connect to future states.

when a violation occurs. An example of the correspondence between the agent's mental state and the world state is shown in Figure 5.1. The conditional obligation is specified in the initial state, and is activated once the condition holds. The figure illustrates that the world semantics may branch into several different states, but only once the state actually changes will the agent configuration change.

5.3 Transition Rules

The semantics allow us to find applicable reasoning rules and use them to update the agent's knowledge bases. We now introduce the transition rules that can be used to make computation runs for AORTA-agents. Each transition rule is only applicable given certain conditions, and by firing a rule, the agent's configuration will change. This makes it possible to compute different traces given an initial state. Transition rules are of the form

$$\frac{\text{premises}}{s \rightarrow s'}$$

where s and s' are agent configurations. If the premises hold, the rule can be executed and the configuration will change. In the following, we include only the parts of the configuration that are used or updated by the transition rules.

It is the case for all rules below, that they are only applicable of the addition or removal of predicates actually change the configuration. That is, adding a role as an enactment option is only possible, if it is not already an option. To make the rules more easily legible we omit this check in the definitions. The rules can easily be expanded to conform to the constraint following the general patterns, where X refers to a set in the agent's configuration:

$$\frac{\ldots \quad \phi \notin X}{X \rightarrow X \cup \{\phi\}} \qquad \frac{\ldots \quad \phi \in X}{X \rightarrow X \setminus \{\phi\}}$$

Furthermore, during an execution, if a rule is not applicable, and it is applied anyway, the operation succeeds without changing the agent configuration.

Unless otherwise stated, we let A refer to an arbitrary agent's name. We do this to ensure that, e.g., the norms rules will activate norm for other role-enacting agents as well as the agent itself. This makes it possible for the agent to reason about norm fulfillment and violation of norms by other agents, allowing the agents to sanction each other.

5.3.1 Norm rules

The norm rules handle the NC-phase of the AORTA-component and connect the norms of the organization with the agent. That is, they update the agent's mental state based on the life cycle of obligations and prohibitions. We distinguish between obligation rules and prohibition rules, and for each type, we define three rules: activation, fulfillment (expiration, in the case of prohibition), and violation. The intention of the rules are based on the definitions in Chapter 4, but are specified using the semantics of AORTA-agents as defined in this character. When applied on, e.g., a conditional norm, the organizational knowledge base is updated with a proper instantiation of the norm and viol predicates of the organizational metamodel.

An conditional obligation is activated for a role-enacting agent, when the condition follows from the current state. In that case, the norm is added to the organizational knowledge base, making it available in the OG- and AE-phases.

Transition rule 5.13 (Obligation activation)**.**

$$\frac{MS \models \mathbf{org}(\mathsf{rea}(A,R)) \wedge \mathbf{org}(\mathsf{cond}(R, obliged, O, D, C)) \wedge C \wedge \neg O}{\Sigma_o \rightarrow \Sigma_o \cup \{\mathsf{norm}(A, R, obliged, O, D)\}} \qquad \text{(Obl-Act)}$$

Note that O,D and C are reasoning formulas.

An obligation has been fulfilled exactly when the obligation can never be violated. In that case, the norm is removed from the organizational knowledge base, since the agent should no longer act upon it in the remaining phases.

Transition rule 5.14 (Obligation fulfillment)**.**

$$\frac{MS \models \mathbf{org}(\mathsf{norm}(A,R,obliged,O,D)) \wedge O}{\Sigma_o \rightarrow \Sigma_o \setminus \{\mathsf{norm}(A,R,obliged,O,D)\}} \quad \text{(Obl-Sat)}$$

Note that O and D are reasoning formulas.

Finally, when an obligation has been violated, it is forever violated. In that case, a viol predicate is added to the organizational knowledge base. No other rules add or remove violations from Σ_o, which means that once an obligation has been violated, the agent(s) can always reason about this (and sanction accordingly).

Transition rule 5.15 (Obligation violation)**.**

$$\frac{MS \models \mathbf{org}(\mathsf{norm}(A,R,obliged,O,D)) \wedge \neg O \wedge D}{\Sigma_o \rightarrow \Sigma_o \cup \{\mathsf{viol}(A,R,obliged,O)\}} \quad \text{(Obl-Viol)}$$

Note that O and D are reasoning formulas.

We can define similar transition rules for prohibitions:

Transition rule 5.16 (Prohibition activation)**.**

$$\frac{MS \models \mathbf{org}(\mathsf{rea}(A,R)) \wedge \mathbf{org}(\mathsf{cond}(R,forbidden,O,D,C)) \wedge C}{\Sigma_o \rightarrow \Sigma_o \cup \{\mathsf{norm}(A,R,forbidden,O,D)\}} \quad \text{(Pro-Act)}$$

Note that O,D and C are reasoning formulas.

Transition rule 5.17 (Prohibition expiration)**.**

$$\frac{MS \models \mathbf{org}(\mathsf{norm}(A,R,forbidden,O,D)) \wedge \neg O \wedge D}{\Sigma_o \rightarrow \Sigma_o \setminus \{\mathsf{norm}(A,R,forbidden,O,D)\}} \quad \text{(Pro-Exp)}$$

Note that O and D are reasoning formulas.

Transition rule 5.18 (Prohibition violation)**.**

$$\frac{MS \models \mathbf{org}(\mathsf{norm}(A,R,forbidden,O,D)) \wedge O}{\Sigma_o \rightarrow \Sigma_o \cup \{\mathsf{viol}(A,R,forbidden,O)\}} \quad \text{(Pro-Viol)}$$

Note that O and D are reasoning formulas.

We abbreviate the execution of norm rules into a single rule, *NC*. We denote choice by | (vertical bar), meaning that one of the rules should be (nondeterministically) chosen for execution, and we let *Rule*$*$ denote that *Rule* is executed until the configuration no longer changes.

Definition 5.19 (Norm check)**.**

$$NC ::= (\textit{Obl-Act}|\textit{Obl-Sat}|\textit{Obl-Viol}|\textit{Pro-Act}|\textit{Pro-Exp}|\textit{Pro-Viol})*$$

5.3.2 Option rules

The option rules correspond to the organizational aspects listed in Section 4.3.2. Here, we let A denote the agent executing the rules.

We let the agent consider a role, if it is capable of achieving at least one main objective of the role.

Transition rule 5.20 (Enactment option)**.**

$$\frac{MS \models \mathbf{org}(\mathsf{role}(R, Os)) \wedge \neg\mathbf{org}(\mathsf{rea}(A, R)) \quad \mathsf{cap}(A) \cap Os \neq \emptyset}{\Gamma_o \rightarrow \Gamma_o \cup \{\mathsf{role}(R)\}} \qquad \text{(Enact)}$$

If all main objectives of an enacting role have been completed, the agent should consider whether deacting that role.

Transition rule 5.21 (Deactment option)**.**

$$\frac{MS \models \mathbf{org}(\mathsf{role}(R, Os)) \wedge \mathbf{org}(\mathsf{rea}(A, R)) \quad Os \subseteq \Sigma_a}{\Gamma_o \rightarrow \Gamma_o \cup \{\neg\mathsf{role}(R)\}} \qquad \text{(Deact)}$$

Active obligations concerning an objective should be considered as options. Only the objective of the obligation is part of the option, but the agent can still reason about the deadline and role using reasoning formulas in the context of action rules.

Transition rule 5.22 (Objective option)**.**

$$\frac{MS \models \mathbf{org}(\mathsf{norm}(A, R, \textit{obliged}, O, D)) \wedge \mathbf{org}(\mathsf{obj}(O, S))}{\Gamma_o \rightarrow \Gamma_o \cup \{\mathsf{obj}(O)\}} \qquad \text{(Objective)}$$

We furthermore generate options for the agents to react to activated and violated norms. While this means that objectives are represented twice in the options-database, it allows for different kinds of rules. E.g., to commit to an objective without considering the deadline, the Objective-rule can be used, otherwise the Norm-rule should be used. Furthermore, we add violation options for *other* agents in the system as well, since it makes sense to be able to reason about these to be able to sanction properly.

Transition rule 5.23 (Norm option)**.**

$$\frac{MS \models \mathbf{org}(\mathsf{norm}(A,R,Deon,O,D))}{\Gamma_o \rightarrow \Gamma_o \cup \{\mathsf{norm}(R,Deon,O,D)\}} \quad \text{(Norm)}$$

Transition rule 5.24 (Violation option)**.**

$$\frac{MS \models \mathbf{org}(\mathsf{viol}(A,R,Deon,O))}{\Gamma_o \rightarrow \Gamma_o \cup \{\mathsf{viol}(A,R,Deon,O)\}} \quad \text{(Violation)}$$

where A is can denote any agent $A \in Ag$ in the system.

The final option rules are concerned with dependency relations. An agent should consider delegating (or requesting the completion of) an objective (by sending an *achieve*-message, if it is in a dependency relation that enables it to do so.

Transition rule 5.25 (Delegation option)**.**

$$\frac{MS \models \mathbf{org}(\mathsf{dep}(R_1,R_2,O)) \wedge \mathbf{org}(\mathsf{rea}(A,R_1)) \wedge \mathbf{opt}(\mathsf{obj}(O))}{\Gamma_o \rightarrow \Gamma_o \cup \{\mathsf{send}(R_2,achieve,O)\}} \quad \text{(Delegate)}$$

Note that O is a reasoning formula, usually $\mathbf{bel}(O')$.

The agent should consider informing about the completion of an objective (by sending a *tell*-message), if other agents depend on that objective.

Transition rule 5.26 (Information option)**.**

$$\frac{MS \models \mathbf{org}(\mathsf{dep}(R_1,R_2,O)) \wedge \mathbf{org}(\mathsf{rea}(A,R_2)) \wedge O}{\Gamma_o \rightarrow \Gamma_o \cup \{\mathsf{send}(R_1,tell,O)\}} \quad \text{(Inform)}$$

Note that O is a reasoning formula, usually $\mathbf{bel}(O')$.

We abbreviate the execution of option rules into a single rule, *OG*, which executes the option rules until none of them are applicable.

Definition 5.27 (Option generation)**.**

$$OG ::= (Enact|Deact|Objective|Norm|Violation|Delegate|Inform)*$$

5.3.3 Action rules

The action transition rules are applicable if there is an option for which the context is entailed by the mental state and the action is defined by the transition function. We distinguish between executing actions and sending a message.

The action execution rule uses an action transition function to change the mental state, only if the context for an option holds. Given that the transition function is partial, the rule is only applicable if the transition function is defined.

Transition rule 5.28 (Executing actions)**.**

$$\frac{o : ctx \rightarrow a \in R_A \quad MS \models \mathbf{opt}(o) \wedge ctx \quad \mathscr{A}(a, MS) = MS'}{MS \rightarrow MS'} \qquad \text{(Act-Exec)}$$

Sending a message is possible when the context holds; the agent adds the message to its outgoing mailbox. Furthermore, we add the fact that the agent has sent the message to the belief base, so that we can specify in the context to only send a certain message once.

Transition rule 5.29 (Sending messages)**.**

$$\frac{o : ctx \rightarrow \mathsf{send}(A, M) \in R_A \quad MS \models \mathbf{opt}(o) \wedge ctx}{\langle \Sigma_a, \mu_{out} \rangle \rightarrow \langle \Sigma_a \cup \{\mathsf{sent}(A, M)\}, \mu_{out} \cup \{\mathsf{msg}(A, M)\} \rangle} \qquad \text{(Act-Send)}$$

where A is the intended recipient of the message, M.

We let action execution, denoted *AE*, be either the execution of an action or sending a message.

Definition 5.30 (Action execution)**.**

$$AE ::= (Act\text{-}Exec|Act\text{-}Send)$$

5.3.4 Other rules

Finally, we define rules for handling external changes and for checking for incoming messages. For handling external changes, we propose a very simple rule with an empty premise. The aim of the operational semantics is to define the interaction between the agent and the organization in terms of an organizational reasoning component. We therefore argue that the interaction between AORTA, different cognitive agents and the environment is not in the scope of this chapter, and we thus abstract away from dealing with the environment. The interaction between AORTA and a cognitive agent can be described using a pair of configurations; one for AORTA and one for the agent. For a deeper discussion of how to deal with these issues, we refer to [Ranathunga et al., 2012].

Transition rule 5.31 (External update)**.**

$$\frac{}{MS \rightarrow MS'} \quad \text{(Ext)}$$

Incoming messages are handled by processing the message through a message transition function, which adds the message to the appropriate knowledge base.

Transition rule 5.32 (Receive messages)**.**

$$\frac{\mathsf{msg}(A, M) \in \mu_{in} \quad \mathscr{M}(A, M, MS) = MS'}{\begin{array}{c}\mu_{in} \rightarrow \mu_{in} \setminus \{\mathsf{msg}(A, M)\} \\ MS \rightarrow MS'\end{array}} \quad \text{(Check)}$$

where A has sent the message, M.

$\mathscr{M}$ is the message transition function, which, as mentioned, may or may not filter incoming messages based on e.g. reputation.

The execution of an entire organizational cycle, *Org*, will then check for messages and external changes, and then execute each of the phases. We let ; (semicolon) denote sequence, i.e., execute the rules delimited by a semicolon in a sequence.

Definition 5.33 (Organizational cycle execution)**.**

$$Org \ ::= \ Check*; Ext; NC; OG; AE$$

We have chosen the cycle as above, but it is possible to define other, e.g., by only executing the NC-phase if, e.g., the environment has changed (via the Ext-rule). We decided to execute all phases in each cycle, since this makes it possible to not only refer to beliefs

in norms, but also organizational concepts and options. For example, we can specify a conditional norm:

$$O_{R_1}(\mathbf{org}(\mathsf{rea}(A, R_2)) < D \mid \mathbf{opt}(\mathsf{role}(R_2))),$$

which states that agents enacting role R_1 is obliged to enact role R_2 before some deadline D if that role becomes an option. This requires us to continuously check whether any updates to the agent's mental state has changed the state of a norm.

Finally, we can define the semantics of AORTA-agents as computation runs using the transition system.

Definition 5.34 (AORTA-agent semantics)**.** The semantics of an AORTA-agent is the set of all computation runs for the agent. A computation run is a sequence, $s_0, \ldots, s_n$ or $s_0, \ldots,$ such that each s_i is a configuration, s_0 is the initial configuration, and for all $s_i, i > 0$, we have that a transition $s_i \longrightarrow s_{i+1}$ can be made in the transition system. For finite computation runs, $s_0, \ldots, s_n$, we have that for s_n there is no s_{n+1} such that $s_n \longrightarrow s_{n+1}$.

Remark (*Agent communication*)**.** One of our key assumptions is that the component will be integrated with cognitive agents that can *communicate* with each other. Furthermore, we assume communication happens via messaging using an in- and outbox. We defined a transition rule for checking messages (i.e., the Check-rule), which takes messages from the inbox, and the send-action, which adds messages to the outbox. It follows from our assumption that the agent will be executed on an agent platform that supports transferring messages between the connected agents.

The communication (i.e., interaction) between the agents in a MAS can be formally specified. Note that we do not describe how agent communication *must* happen, but rather show one approach to it (based on the approach by Tinnemeier [2011], where labeled transitions denote communication between components).

We define a MAS configuration as set of agents:

Definition 5.35 (Multi-agent system configuration)**.** A multi-agent system configuration is a tuple:

$$MAS = \langle \mathbb{A} \rangle,$$

where $\mathbb{A}$ is the set of agents in the system.

When an agent adds a message to the outbox (using the send-action), it is subsequently sent to the recipient. This is defined by the following rule.

Transition rule 5.36 (Agent communication)**.**

$$\frac{A_1 \xrightarrow{\mathsf{msg}(\alpha_2,msg)!} A_1' \quad A_2 \xrightarrow{\mathsf{msg}(\alpha_1,msg)?} A_2'}{\langle \mathbb{A} \cup \{A_1, A_2\}\rangle \rightarrow \langle \mathbb{A} \cup \{A_1', A_2'\}\rangle}, \qquad \text{(Agt-Msg)}$$

where A_1 and A_2 are agent configurations, α_1 and α_2 corresponds to the name of A_1 and A_2, respectively. The configurations are updated such that $A_1' = A_1$ with $\mathsf{msg}(\alpha_2, msg)$ removed from the outbox and $A_2' = A_2$ with $\mathsf{msg}(\alpha_1, msg)$ added to the inbox.

5.4 Executing AORTA-agents

In this section, we show an execution trace of four agents in the EAH scenario. The complete organizational metamodel for the example is shown in Table 5.3, the action rules (available to all agents) are shown in Table 5.4, and the execution traces are shown in Figure 5.5. It is a rather large example, which may seem overwhelming, so we describe a number of selected parts in detail.

We focus on the organizational reasoning and are not concerned with *how* the agents complete their goals (e.g., how to put an item up for sale or verify a customer).

Example 5.37 (PH-lamp)**.** *Sally wants to sell her PH-lamp using the EAH. Both Bob and Carol are interested in buying it.*

For some reason, Bob is known to provide incorrect information when registering for various services. For this reason, he should not be allowed to be verified, and since he knows this, he has to circumvent the rules in order to buy the lamp. He does so by simply bidding on the lamp without waiting to be verified, but since this violates a norm, the manager, Mike, chooses to sanction him by banning him from the auction.

Carol wins the lamp and is obliged to pay for it, but violates this by leaving the auction without paying. She then reacts to the violation by paying for the item and avoids sanctions.

The initial state of each agent is shown in Table 5.2.

Table 5.2: *Initial state of each agent in the EAH scenario.*

	BOB	**CAROL**	**SALLY**	**MIKE**
bel	*want*(“PH-lamp”) *badInfo*(*bob*)	*want*(“PH-lamp”) *badInfo*(*bob*)	*have*(“PH-lamp”) *badInfo*(*bob*)	*badInfo*(*bob*)
goal	*bought*(“PH-lamp”)	*bought*(“PH-lamp”)	*sold*(“PH-lamp”)	
cap	*registered*(*Ag*) *bought*(*Item*) *bid*(*Ag*,*Item*)	*registered*(*Ag*) *bought*(*Item*) *bid*(*Ag*,*Item*)	*registered*(*Ag*) *sold*(*Item*) *auction*(*Ag*,*Item*)	*verified*(*Ag*)

Table 5.3: Organizational metamodel for the EAH scenario. For clarity, we omit the type of reasoning formula for each formula inside the predicates (e.g., we write *registered*(*Agent*) rather than **bel**(*registered*(*Agent*))).

Roles

MM1: role(*customer*, [*registered*(*Agent*), *bought*(*Item*), *sold*(*Item*), *paid*(*Item*)])

MM2: role(*buyer*, [*bid*(*Agent*, *Item*), *verified*(*Agent*)])

MM3: role(*seller*, [*auction*(*Agent*, *Item*), *verified*(*Agent*)])

MM4: role(*manager*, [*verified*(*Agent*)])

Objectives

MM5: obj(*registered*(*Agent*), [])

MM6: obj(*verified*(*Agent*), [*registered*(*Agent*)])

MM7: obj(*bought*(*Item*), [*bid*(*Agent*, *Item*), *paid*(*Item*)])

MM8: obj(*sold*(*Item*), [*auction*(*Agent*, *Item*)])

MM9: obj(*paid*(*Item*), [])

MM10: obj(*bid*(*Agent*, *Item*), [])

MM11: obj(*auction*(*Agent*, *Item*), [])

Dependencies

MM12: dep(*customer*, *manager*, *verified*(*Agent*))

Conditional norms

MM13: cond(*customer*, *obliged*, *registered*(*Me*), *verified*(*Me*), *agent*(*Me*))

MM14: cond(*customer*, *obliged*, *paidFor*(*Item*),
¬*participates*(*Me*, *Item*), *won*(*Me*, *Item*))

MM15: cond(*buyer*, *obliged*, *verified*(*Me*),
bid(*Me*, *Item*), *registered*(*Me*))

MM16: cond(*seller*, *obliged*, *verified*(*Me*),
auction(*Me*, *Item*), *registered*(*Me*))

MM17: cond(*manager*, *forbidden*, *verified*(*Agent*), ⊥,
badInfo(*Agent*) ∧ *registered*(*Agent*))

Table 5.4: Action rules for the auction house agents. We assume the variable *Me* binds to the agent's own name.

Role enactment

AR1: $\mathsf{role}(customer) \ : \ \top \rightarrow \mathsf{enact}(customer)$

AR2: $\mathsf{role}(buyer) \ : \ \mathbf{goal}(bought(Item)) \wedge \mathbf{bel}(registered(Me)) \rightarrow \mathsf{enact}(buyer)$

AR3: $\mathsf{role}(seller) \ : \ \mathbf{goal}(sold(Item)) \wedge \mathbf{bel}(registered(Me)) \rightarrow \mathsf{enact}(seller)$

AR4: $\mathsf{role}(manager) \ : \ \top \rightarrow \mathsf{enact}(manager)$

Coordination

AR5: $\mathsf{send}(\top, tell, \mathbf{org}(\mathsf{rea}(Me, Role)))$
$: \ \mathbf{bel}(agent(A)) \wedge \neg\mathbf{bel}(A = Me) \wedge \neg\mathbf{bel}(\mathsf{sent}(A, \mathbf{org}(\mathsf{rea}(Me, Role))))$
$\rightarrow \mathsf{send}(A, \mathbf{org}(\mathsf{rea}(Me, Role)))$

AR6: $\mathsf{send}(manager, achieve, \mathbf{bel}(verified(Me)))$
$: \ \mathbf{org}(\mathsf{rea}(A, manager)) \wedge \neg\mathbf{bel}(\mathsf{sent}(A, \mathbf{opt}(\mathsf{obj}(\mathbf{bel}(verified(Me))))))$
$\rightarrow \mathsf{send}(A, \mathbf{opt}(\mathsf{obj}(\mathbf{bel}(verified(Me)))))$

AR7: $\mathsf{send}(customer, tell, \mathbf{bel}(verified(You)))$
$: \ \mathbf{org}(\mathsf{rea}(You, customer)) \wedge \neg\mathbf{bel}(\mathsf{sent}(You, \mathbf{bel}(verified(You))))$
$\rightarrow \mathsf{send}(You, \mathbf{bel}(verified(You)))$

Commitment

AR8: $\mathsf{obj}(\mathbf{bel}(verified(Agent))) \ : \ \mathbf{org}(\mathsf{rea}(Me, Manager))$
$\wedge \neg\mathbf{org}(\mathsf{norm}(Me, manager, forbidden, \mathbf{bel}(verified(agent)), \bot))$
$\rightarrow \mathsf{commit}(verified(Agent))$

AR9: $\mathsf{obj}(\mathbf{bel}(registered(Me))) \ : \ \mathbf{goal}(bought(Item)) \vee \mathbf{goal}(sold(Item))$
$\rightarrow \mathsf{commit}(registered(Me))$

AR10: $\mathsf{viol}(Me, customer, obliged, \mathbf{bel}(paidFor(Item))) \ : \ \mathbf{bel}(want(Item))$
$\rightarrow \mathsf{commit}(paidFor(Item))$

AR11: $\mathsf{viol}(Ag, buyer, obliged, \mathbf{bel}(verified(Ag)))$
$: \ \mathbf{org}(\mathsf{rea}(Me, manager)) \wedge \mathbf{bel}(participates(Ag, Item))$
$\rightarrow \mathsf{commit}(banned(Ag))$

AR12: $\top \ : \ \mathbf{goal}(bought(Item)) \wedge \mathbf{bel}(auction(Ag, Item)) \wedge \mathbf{bel}(registered(Me))$
$\wedge \mathbf{bel}(badInfo(Me)) \wedge \mathbf{bel}(participates(Me, Item))$
$\rightarrow \mathsf{commit}(bid(Item))$

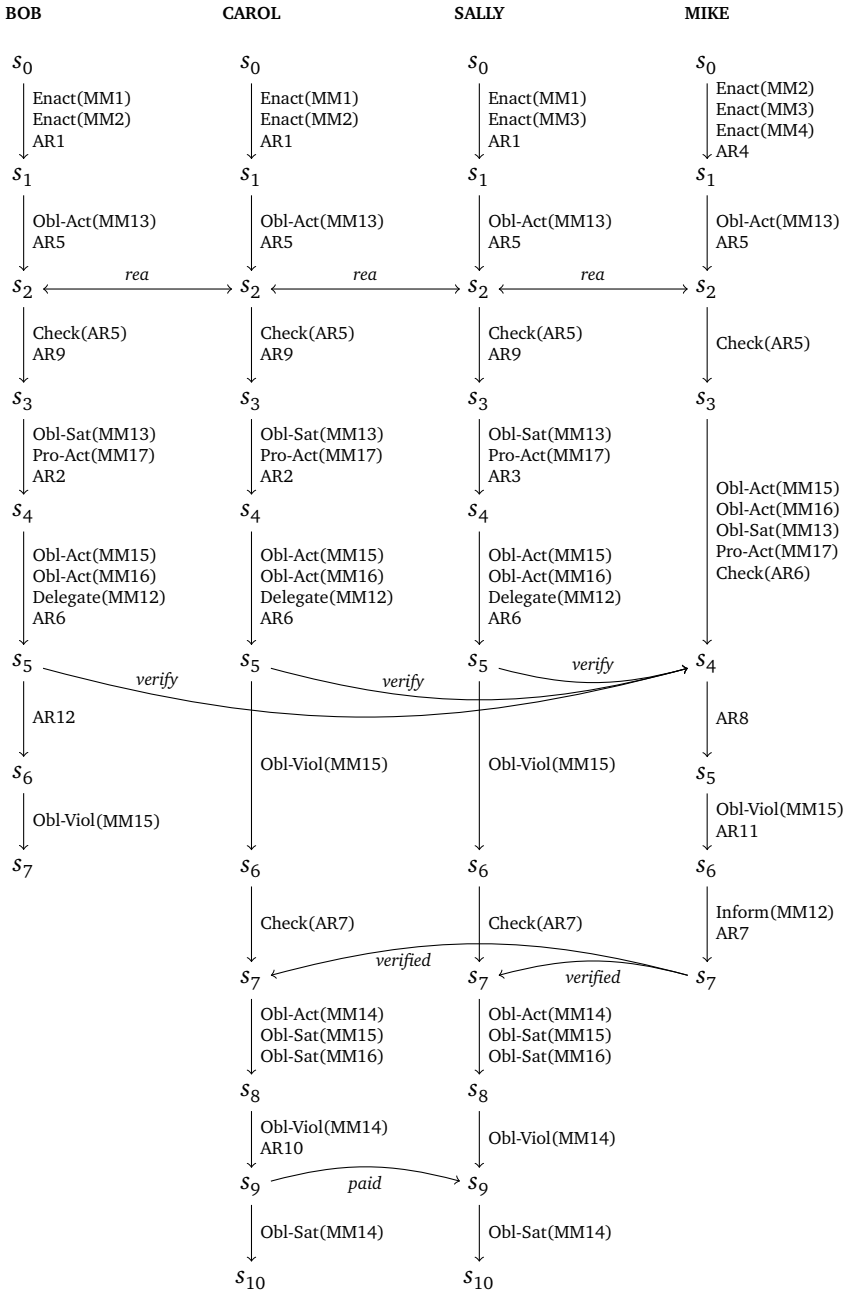


Figure 5.5: Execution of the EAH scenario.

In Figure 5.5, each transition from state s_i to state s_{i+1} is labeled with the transition rules that changed the agent's configuration. We write *Rule*(*MMi*) to indicate that the rule is applicable based on the predicate in the metamodel, where *Rule* is a transition rule and *MMi* refers to predicate *i* in the organizational metamodel[3]. Interaction between the agents is indicated by a bent arrow with a label specifying the kind of interaction (e.g. request to verify or information about role enactment).

Since the execution happens in parallel, we look at groups of state transitions from the different agents. We assume that each agent holds a belief *agent*(*A*) for each agent *A* in the system. Furthermore, we omit certain transitions for clarity (e.g., when Bob enacts the buyer role, he uses AR5 to inform the other agents, but this transition is not shown since we have already seen the same rule in use for informing about the customer role).

Role enactment: From s_0 to s_1, all of the agents generate options to enact roles based on their capabilities. Notice that Mike generates options to enact the buyer and seller roles (since one objective of these roles is to become verified). The execution of AR1 and AR4 enacts the *customer* role and *manager* role, respectively.

Role coordination: The enact-action adds an option to tell other agents about the enactment. AR5 is thus applicable and each agent will execute it for every other agent in the system[4].

Objective coordination: The role coordination means that the customers know who is manager and can use the dependency relation to delegate the verification objective, i.e., Delegate(MM12). When the manager has verified the agents, the dependency relation can be used in reverse to inform about the verification, i.e., Inform(MM12).

Norm activation: When Bob registers, a prohibition to verify his credentials is activated since he is known to provide incorrect information. Thus when Mike verifies the other agents using AR8, the prohibition prevents him from even trying to verify Bob.

Norm violation: Several violations occur: when Bob bids on an item without being verified (using AR12), he violates (the activated version of) the norm MM15.

When Carol leaves the auction before paying (by performing some action not regulated by the organization), she violates MM14. Notice that even though the violation occurred because of a "selfish" action (i.e., one based on her own desires), she can use AR10 to recover from the violation. We do not know why she initially did not pay for the item; perhaps she did not know she had to pay before leaving. In that case, AORTA made it possible to change her course of action and eventually pay for the item.

This concludes the execution of the EAH scenario. The example shows that the agents can use the AORTA-component to enact roles based on their own capabilities and goals

[3] In the case of sending messages, we write *Check*(*ARi*) to indicate applicability based on an action rule to send a message, executed by another agent

[4] For clarity, we omit the extra transitions that would normally occur due to AR5.

and are able to follow the norms of the organization (registering, getting verified, paying for items, etc.).

We have thus shown that by using AORTA the organizational model in the system is made available for the agents to act upon. In the OG-phase, the framework generates options in relation to the model and the state of the system, thus helping the agents decide what to do in relation to the organization. Since the capabilities of the agents are preexistent, AORTA mainly provides a functional way for the agents to use their capabilities in completing the organizational objectives. Furthermore, in the NC-phase, norms are activated for the agents, and are available to not only responsible agents, but other agents as well, giving a way for them to react to fulfillment and violation by other agents in the system. We showed that the manager could decide to ban a customer, if the customer tried to bid on items without being verified. This was possible exactly because the NC-phase handles norms for every agent, not just the agent itself.

How the agents use AORTA is specified by the action rules, and they should be specified by the programmer. While very general rules can be established (e.g. always enact possible roles, always commit to objectives), more specific rules allow the agents to better react to preconditions for performing a task or to possible violations.

The fact that each agent uses a separate AORTA-component means that the agents may hold different beliefs about the organization, just as they may have different views of their environment. The Inform-rule and the enact-action accommodate this by generating send-options for maintaining consistency among the agents. For example, when Mike enacts the manager role, an option is generated to tell other agents in the system about this, and when an agent enacting the customer role should verify her credentials, an option is generated to delegate this to the manager. Using the agent's beliefs about role enactment, this makes it possible for the agents to become verified without having the manager to continuously check for new registrations.

Furthermore, since the component is added to the agent and assumes nothing a priori about that agent, the plans inside the agent may very well conflict with the AORTA action rules. For example, the agent may choose not to commit to an objective using AORTA, but adopt it anyway using its internal plans (e.g., if the manager is a friend of Bob, he might verify his credentials anyway). However, if this leads to a violation of an organizational obligation, other agents will be aware of this, and can choose to punish the agent. Thus, AORTA allows the agents to follow the rules of an organization, but poses no restrictions on them to do so; if they choose to follow their own plans, this cannot be prevented – and rightfully so: otherwise their autonomy would be severely limited.

We return to the example in Chapter 7, where we show how to implement it in the AORTA architecture integrated with various agent platforms.

5.5 Concluding Remarks

We have presented the operational semantics of the AORTA-component. We have discussed how to represent the agent's mental state in order to capture both organizational and world beliefs. We have introduced a number of organizational actions that can be

used by the agent to alter the organizational state or commit to organizational objectives. Furthermore, we have introduced a complete transition system for the component that go through the three phases of the AORTA-component (norm check, option generation and action execution). Finally, we showed how to execute AORTA-agents using the EAH scenario.

PART THREE

ENGINEERING ORGANIZATION-AWARE AGENTS

This part focuses on *building* organization-aware agents. We present the AORTA architecture, a Java-based implementation of the AORTA framework (Chapter 6). We then provide examples of organization-aware agents implemented in AORTA, integrated with existing agent platforms (Chapter 7). Finally, we show how we can use *model checking* to verify agents that use AORTA to perform organizational reasoning using Agent Java PathFinder (Chapter 8).

CHAPTER 6

The AORTA Architecture

In this chapter, we move from theory to practice: having presented the AORTA framework in clear operational semantics, we can now make a practical implementation of the framework and integrate it with existing agent platforms. The chapter is based on [Jensen et al., 2014], but is greatly extended to incorporate an implementation of the complete semantics of AORTA. Furthermore, we present a way to specify the AORTA metamodel to make the design of organization-oriented systems based on AORTA more straightforward and intuitive.

We will present the *AORTA architecture*, which is the implementation of the theoretical framework described in the previous chapters. Since agents using AORTA are based on an organizational metamodel and a set of reasoning rules, the implementation includes languages for specifying the metamodel and AORTA programs. We finally show agent implementations based on the running example.

There exists many definitions of the concept "software architecture". We consider it the highest level of abstraction of a software system and to describe it, we use the definition from the international standard for architecture descriptions of systems and software, "ISO/IEC/IEEE Systems and software engineering – Architecture description" [2011]:

> *(The architecture consists of the) fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution.*

The AORTA architecture is designed as a *standalone* system, which means that it is completely decoupled from the agent platforms it is integrated with. This is done for several reasons:

- *Separation of concerns* (this chapter): a complete decoupling ensures that organizational reasoning is done in the component and that it does not interfere with the agent's normal reasoning cycle.
- *Ease of integration* (Chapter 7): the system contains its own reasoning engine, so integration with an agent platform does not depend on the agent platform's reasoning engine.
- *Verifiability* (Chapter 8): the AJPF verification tool is tailored for MAS, and a standalone reasoning framework can be integrated with this tool similarly to agent platforms.

We have implemented the framework as described in this chapter using Java SE 7. The choice of Java was easy: the most popular existing agent platforms (including *Jason*, GOAL and 2APL) are implemented in Java, so for integration purposes, Java is a natural

choice. Furthermore, the IDE for OperA, *OperettA* [Aldewereld and Dignum, 2011], and the OperA model itself, is also implemented in Java, so to implement the translation process from OperA to the metamodel, Java was needed. We have released AORTA as an Open Source project[1] allowing others to contribute as well.

Due to the complete decoupling of AORTA with agent frameworks, AORTA comes with its own reasoning engine. Inspired by existing agent platforms, and in order to integrate the metamodel in the most intuitive way, our choice was to use Prolog as reasoning engine. The implementation of the reasoning engine uses tuProlog [Denti et al., 2001], which is a Java-based lightweight implementation of ISO-Prolog. We chose tuProlog because of its efficiency and straightforward interface in Java, allowing us to query a Prolog knowledge base without requiring any external system-dependent libraries. This effectively makes AORTA standalone, but as we shall see, it puts a synchronization requirement on the integration with existing agent platforms.

6.1 The AORTA Metamodel

The metamodel is intentionally designed to consist of a number of predefined predicates that agents can refer to using their reasoning rules. As described in Chapter 5, this makes the semantics of the mental state straightforward, and lets us easily refer to the metamodel in the transition rules. From a programming point of view, however, specifying the metamodel as a set of predicates is not ideal. First, redundancy of information could be a problem regarding legibility, since, e.g., for every role we have to specify `role(...)`, and second, one has to carefully make sure that each predicate has the correct number of arguments, and so on. In smaller projects, this may not be a problem, but as soon as the model contains more than a few roles, objectives and obligations, the specification may become difficult to read and edit.

We thus propose a simple language, which divides the model into five sections: *roles*, *objectives*, *dependencies*, *conditional norms* and *rules*. The syntax for specifying the metamodel is shown in Table 6.1.

We explain the concepts of the metamodel using an excerpt of metamodel for the EAH scenario shown in Table 6.2. Roles are identified by their name and each consists of an optional list of objectives. Objectives are identified by their state description and optionally sub-objectives. Dependencies are represented by the roles and the objective in the relation. Conditional norms are as states that, given a condition, are obligatory or prohibited for a role before some other event occurs.

Notice the optional variable `Me` in line 13, which is used to decide for which agent(s) the condition must hold. The variable ensures that the norm is only activated for agents where `registered(Me)` holds with `Me` unified with their *own name*. Otherwise, we might have a situation where, e.g., Bob is obliged to getting Carol verified, since she is registered as well. That is, if Bob believes `registered(bob)` and `registered(carol)`, he would (wrongly) be obliged to the following norms:

[1] AORTA is available at `https://github.com/andreasschmidtjensen/aorta`.

Table 6.1: Extended Backus-Naur Form for the AORTA Metamodel.

⟨*metamodel*⟩	::=	⟨*roles*⟩ ⟨*objectives*⟩ ⟨*dependencies*⟩? ⟨*norms*⟩? ⟨*rules*⟩?
⟨*roles*⟩	::=	'ROLES:' ⟨*role*⟩+
⟨*role*⟩	::=	⟨*identifier*⟩ '.' | ⟨*identifier*⟩':' ⟨*termlist*⟩ '.'
⟨*objectives*⟩	::=	'OBJECTIVES:' ⟨*objective*⟩+
⟨*objective*⟩	::=	⟨*term*⟩ '.' | ⟨*term*⟩':' ⟨*termlist*⟩ '.'
⟨*dependencies*⟩	::=	'DEPENDENCIES:' ⟨*dependency*⟩+
⟨*dependency*⟩	::=	⟨*identifier*⟩ '>' ⟨*identifier*⟩ ':' ⟨*term*⟩ '.'
⟨*norms*⟩	::=	'NORMS:' ⟨*norm*⟩+
⟨*norm*⟩	::=	⟨*identifier*⟩ {'='⟨*var*⟩}? '[' ⟨*deon*⟩ ']':' ⟨*term*⟩ '<' ⟨*term*⟩ '|' ⟨*term*⟩ '.'
⟨*deon*⟩	::=	'obliged' | 'forbidden'
⟨*rules*⟩	::=	'RULES:' ⟨*rule*⟩+
⟨*rule*⟩	::=	any legal Prolog clause '.'
⟨*termlist*⟩	::=	⟨*term*⟩ {',' ⟨*term*⟩}+
⟨*term*⟩	::=	any legal Prolog term
⟨*identifier*⟩	::=	any legal Prolog atom
⟨*var*⟩	::=	any legal Prolog variable

Table 6.2: Excerpt from the metamodel of the EAH scenario.

```
1  ROLES:
2  customer: registered(Agent); bought(Item); sold(Item); paid(Item).
3  manager: verified(Agent).
4
5  OBJECTIVES:
6  registered(Agent).
7  verified(Agent): registered(Agent).
8
9  DEPENDENCIES:
10 customer > manager: verified(Agent).
11
12 NORMS:
13 customer=Me [obliged]: registered(Me) < verified(Me) | agent(Me).
14 manager [forbidden]: verified(Ag) < false | badInfo(Ag), registered(Ag).
15
16 RULES:
17 registered(Agent) :- registered(Agent, Address, Account).
```

```
norm(bob, obliged, verified(carol), bid(carol, Item)).
  norm(bob, obliged, verified(bob), bid(bob, Item)).
```

We can also have situations where a norm is *required* to take other agents' situation into account (see line 14). Here, the manager is prohibited from verifying an agent if that agent is known to provide bad information and has registered. Since this is only possible if the agent can consider other agents, we do not want a binding between the role-enacting agent and the agent considered in this norm.

Finally, rules are added for convenience and allow the programmer to specify Prolog clauses that can be referred to elsewhere in the metamodel and can be used by the agents in their reasoning. We can view the rules as a simple kind of *counts-as* rules. For example, the rule in line 17 specifies that registering with name, address and account information *counts as* registering in the EAH domain.

The full metamodel of the EAH can be found in Appendix A.1.

6.2 AORTA-programs

The AORTA-program specifies how an agent reacts to the options generated in the OG-phase. Following the operational semantics of AORTA, a program consists of a list of action rules of the form

```
option : context => action.
```

Options include role enactment; considering objectives, norms and violations; and sending messages. The context describes the state in which the rule is applicable and is specified by a reasoning formula. The action specifies what should be executed. The complete syntax of AORTA-programs is shown in Table 6.3.

Table 6.3: Extended Backus-Naur Form for AORTA programs.

⟨*program*⟩	::=	⟨*rule*⟩+
⟨*rule*⟩	::=	⟨*if_rule*⟩
	|	⟨*action_rule*⟩
⟨*if_rule*⟩	::=	'if' ⟨*context*⟩ '{' ⟨*rule*⟩+ '}'
⟨*action_rule*⟩	::=	⟨*option*⟩ ':' ⟨*context*⟩ '=>' ⟨*action*⟩ '.'
⟨*option*⟩	::=	{'~'}? 'role(' ⟨*identifier*⟩ ')'
	|	{'~'}? 'obj(' ⟨*term*⟩ ')'
	|	'norm(' ⟨*term*⟩ ',' ⟨*deon*⟩ ',' ⟨*term*⟩ ',' ⟨*term*⟩ ')'
	|	'viol(' ⟨*term*⟩ ',' ⟨*term*⟩ ',' ⟨*deon*⟩ ',' ⟨*term*⟩ ')'
	|	'send(' ⟨*identifier*⟩ ',' ⟨*ill_force*⟩ ',' ⟨*term*⟩ ')'
	|	'true'
⟨*deon*⟩	::=	'obliged' | 'forbidden'
⟨*ill_force*⟩	::=	'tell' | 'achieve'
⟨*action*⟩	::=	'enact(' ⟨*identifier*⟩ ')'
	|	'deact(' ⟨*identifier*⟩ ')'
	|	'commit(' ⟨*term*⟩ ')'
	|	'drop(' ⟨*term*⟩ ')'
	|	'send(' ⟨*identifier*⟩ ',' ⟨*term*⟩ ')'
⟨*context*⟩	::=	⟨*formula*⟩ { ',' ⟨*formula*⟩ }+
	|	'true'
⟨*formula*⟩	::=	{'~'}? 'org(' ⟨*termlist*⟩ ')'
	|	{'~'}? 'opt(' ⟨*termlist*⟩ ')'
	|	{'~'}? 'bel(' ⟨*termlist*⟩ ')'
	|	{'~'}? 'goal(' ⟨*termlist*⟩ ')'
	|	{'~'}? 'cap(' ⟨*term*⟩ ')'
⟨*termlist*⟩	::=	⟨*term*⟩ {',' ⟨*term*⟩}+
⟨*term*⟩	::=	any legal Prolog term
⟨*identifier*⟩	::=	any legal Prolog atom

Table 6.4: Excerpt from the AORTA-program for the EAH scenario.

```
if bel(me(Me)) {
  role(customer) : true => enact(customer).
  role(manager) : cap(banned(Agent)) => enact(manager).

  if org(rea(Me, manager)) {
    obj(bel(verified(Agent)))
      : ~org(norm(Me, manager, forbidden, bel(verified(Agent)), false))
      => commit(verified(Agent)).
    viol(Ag, buyer, obliged, bel(verified(Ag)))
      : bel(participates(Ag,_)) => commit(banned(Ag)).
  }

  obj(bel(registered(Me))) : goal(bought(Item))
    => commit(registered(Me)).
  obj(bel(registered(Me))) : goal(sold(Item))
    => commit(registered(Me)).
}
```

An excerpt of an AORTA-program is shown in Table 6.4[2]. Notice that there are two additions to the syntax presented here compared to the operational semantics: *if-rules* and reasoning about *capabilities*.

If-rules are introduced because we have experienced that certain parts of the context is often reused in many parts of a program. For example, a number of rules might only be applicable while the agent is enacting the *manager* role, which means the context of every rule in that set will contain $\mathbf{bel}(me(Me)) \wedge \mathbf{org}(\mathrm{rea}(Me, manager))$, where $me(Me)$ is a predicate that succeeds if *Me* evaluates to the agent's name. This makes the rules unnecessarily long and hard to read, since the programmer has to process a lot of redundant information.

If-rules are thus added as *syntactic sugar* in the implementation of the framework.

Definition 6.1 (If-rule)**.** An if-rule in an AORTA-program is a rule of the form

```
if context { rules }
```

where `rules` is a set of rules (action or if-rules), and `context` describes a state in which the rules should be (attempted to be) executed.

Since if-rules are syntactic sugar, a program using if-rules is in reality transformed into a program without if-rules. That is, given an AORTA-program,

```
if context1 { option : context2 => action }
```

we can transform the program into an equivalent program without if-rules by changing the context of all rules inside the if-rule to a conjunction of their current context and the if-context:

[2]The full AORTA-program for the EAH can be found in Appendix A.2.

```
option : context1, context2 => action
```

where comma (,) denotes conjunction, since Prolog is used as reasoning engine. The programs are equivalent, in the sense that the outcome is the same: if no solution can be found for `context1`, then no solution can be found for `context1, context2`. Therefore, if the if-rule is not applicable, the transformed action-rule would not be applicable.

Example 6.2 (Transformation of programs with if-rules)**.** *The following program is a transformed version of the rules in Table 6.4.*

```
role(customer) : bel(me(Me)) => enact(customer).
role(manager) : bel(me(Me)), cap(banned(Agent)) => enact(manager).

obj(bel(verified(Agent))) : bel(me(Me)), org(rea(Me, manager)),
  ~org(norm(Me, manager, forbidden, bel(verified(Agent)), false))
  => commit(verified(Agent)).
viol(Ag, buyer, obliged, bel(verified(Ag)))
  : bel(me(Me)), org(rea(Me, manager)), bel(participates(Ag,_))
  => commit(banned(Ag)).

obj(bel(registered(Me))) : bel(me(Me)), goal(bought(Item))
  => commit(registered(Me)).
obj(bel(registered(Me))) : bel(me(Me)), goal(sold(Item))
  => commit(registered(Me)).
```

Even though the two programs are equivalent, we find that the use of if-rules clearly enhances legibility.

We allow the use of cap(ϕ) in the rule context to reason about capabilities, e.g., as in the example to only enact the manager role if the agent is able to ban agents from the auction. While this is not part of the operational semantics, we allow the programmer to access the capabilities, simply because they are located inside the Prolog database containing the mental state, and are thus directly available for querying. This not only allows us to use capabilities to generate options, but also for deciding which rules to execute.

6.3 AORTA-component

The AORTA-component is designed to be as faithful as possible to the agent configuration in Chapter 5. The agent is thus identified by its name, mental state, action rules, transition functions, capabilities and mailbox. Furthermore, we introduce the notion of an *external agent*, which is used to synchronize the Prolog knowledge base with the mental state of the cognitive agent. In the following, we describe our design choices for each part and finally, we show how they, when assembled, result in a functional AORTA-component that can be integrated in existing agents.

6.3.1 Mental state

The mental state contains the agent's beliefs, goals, organizational beliefs and options. In order to operationalize the action rules, we need to be able to both query and alter the

mental state. We accommodate this by letting the mental state consist of a Prolog database. In order to keep the distinction between the different knowledge bases in AORTA, we use an unary predicate for each rule to indicate its type. For example, we can make the following conversion of the context in a rule:

```
bel(a, b), ~org(c, d) ⟹ bel(a), bel(b), \+ (org(c), org(d))
```

This makes querying straightforward, while still keeping the distinction between the different knowledge bases.

In the definition of the organizational belief base, we limit the allowed terms to those of the metamodel. In the implementation, there is no explicit limitation put on the terms that can be added to the mental state, but it is effectuated by the transition rules and actions, since the organizational belief base is only altered by these.

Note that the AORTA-component contains its own copy of the agent's mental state, rather than integrating AORTA into the knowledge bases of the agent in an existing platform. This means that the belief base and goal base of AORTA must be synchronized with the cognitive agent, which could lead to pitfalls in an integration process. However, it makes querying easier, since only one query engine is used. Furthermore, our aim is to enable AORTA to be integrated with most of the existing agent platforms, and since it requires only that formulas must be converted between the language of AORTA and the agent platform in question, we find that it makes the implementation of AORTA simpler to understand.

6.3.2 Executing actions

Actions determine what the agent can do. In the context of AORTA, this means role enactment, objective commitment and sending messages. As per the semantics, an action is applicable if the context matches the agent's mental state and the action is defined by the agent's action transition function. We have, however, not explicitly implemented the action transition function, but rather implemented each action as a class that can alter the mental state of the agent. This was done to make the integration of new actions as straightforward as possible.

The message transition function is explicitly implemented with a very simple behavior: each incoming message is simply added to the mental state. We hinted when we described the operational semantics that the agent might use more sophisticated functions to consider, e.g., the sender's trustworthiness. This is indeed possible in the architecture: the `MessageFunction` can be extended to perform preprocessing of incoming messages to achieve such functionality.

6.3.3 Capabilities

The capabilities of an agent describe what the agent is able to achieve. In the context of AgentSpeak-based languages, this is essentially described by the plans with a triggering event of the type *goal achievement*. Capabilities thus depend on the agent platform, but they are used by AORTA for option generation. Therefore, the integration process

includes a part that extracts an agent's capabilities to be used by AORTA. The extracted capabilities are added to the Prolog database as predicates cap(ϕ), where ϕ denotes a capability, effectively making them available to the transition rules.

6.3.4 Mailbox

The operational semantics of AORTA uses an in- and outbox for organizational messages. The Check transition rule processes messages from the inbox, and the send-action adds messages to the outbox. Since the operational semantics handles only the AORTA-component, it does not deal with moving messages back and forth between different agents.

In the implementation, we keep an organizational inbox internal to AORTA. The Check-rule processes messages from this inbox. Outgoing messages are added to an internal organizational outbox, which is processed every cycle by using the agent platform's mail system to send the messages.

6.3.5 External agent

The external agent is a component that receives updates from the cognitive agent, whenever it adds or removes beliefs (or goals) to its belief base (or goal bases). It is designed such that synchronization between the cognitive agent and the AORTA mental state happens only once in each reasoning cycle (namely in the beginning, if we use the built-in strategy). Since we make no assumptions about the underlying agent platform, it might continuously add percepts to the belief base from the environment, so by limiting the synchronization to a single state in the reasoning cycle, we avoid race conditions. We further elaborate on the external agent and other integration issues in Chapter 7.

6.3.6 Reasoning cycle

The reasoning cycle performs two steps: first, the transition rules are executed, and second, any outgoing messages will be processed and sent using the agent platform's mail system.

Transition rules are executed using a strategy, which defines the order in which they are executed. AORTA comes with a built-in strategy, which implements the organizational cycle as explained in Chapter 5. The execution of action rules (i.e., Act-Exec and Act-Send) is performed in a linear manner; the AORTA-program is processed from the top, and the first matching rule is then executed. If no matching rule is found, the agent's state remains unchanged.

Most of the transition rules alter the mental state of the agent. Each of these rules is implemented as a Prolog query, which, if successful, leads to a change in the mental state. For example, the Enact-rule generates options to enact a role using the following query (where A has been replaced by the agent's name):

```
org(role(R,Os)), \+ org(rea(A,R)), \+ opt(role(R,Os)),
   member(G,Os), cap(G).
```

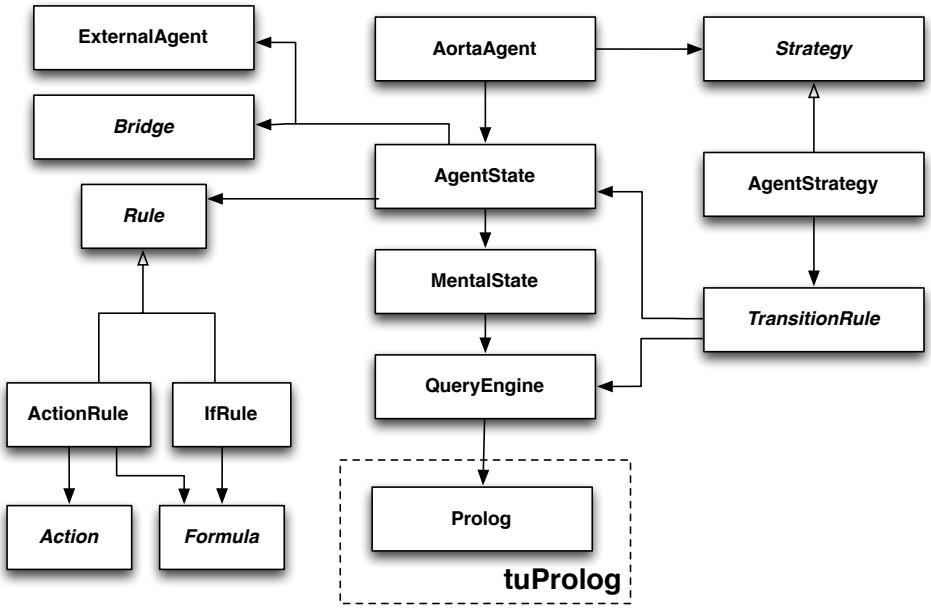


Figure 6.5: Class diagram of the AORTA architecture.

That is, a role is an option if: (1) the agent does not currently enact it, (2) it is not already an option, and (3) the agent is capable of achieving at least one objective of the role. If the query succeeds, an option is added to the mental state:

```
assert(opt(role(R))).
```

Most of the other transition rules are implemented in a similar manner. We briefly explain the rules that stand out. The Act-Exec-rule delegates the execution to an `Action` object, which decides if the action is applicable based on the agent's current configuration. If that is the case, the action is executed, and the mental state is updated. The Act-Send-rule adds messages to the mailbox of the agent. The Check-rule takes an incoming message and processes it using the message transition function, thereby changing the mental state. Finally, the Ext-rule is used to synchronize the mental state with the agent's beliefs and goals, which means it is integrated with the external agent.

6.3.7 Overview

An overview of the parts of the AORTA architecture is shown in Figure 6.5. An agent from some agent platform (e.g. *Jason*) is associated with an instance of `AortaAgent`, which contains AORTA's view of the agent's state, `AgentState`, and in which the reasoning

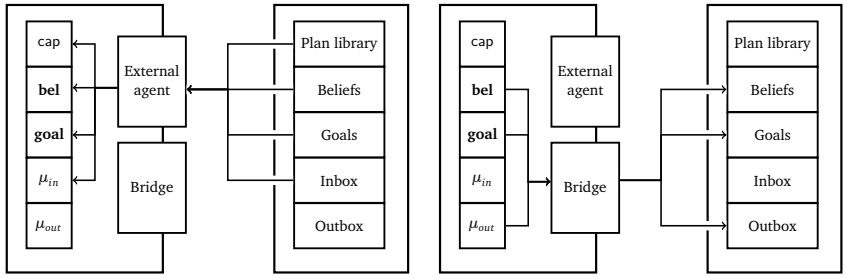
cycle is implemented as a `Strategy`. The mental state is connected to a query engine, which uses tuProlog to perform all reasoning. Furthermore, the agent state is connected to two classes concerned with integration with agent platforms: `ExternalAgent` and `AortaBridge`. These classes are responsible for synchronizing the agent's mental state with the component's view the state. The external agent synchronizes beliefs and goals from the agent to the component, and the bridge synchronizes beliefs and goals from the component to the agent.

6.4 Integration with Cognitive Agents

The design of our component is based on the assumption that it will be connected to a *cognitive agent*, that is, an agent with beliefs, goals and means for achieving the goals (e.g. a plan library). We furthermore require that the beliefs and goals can be represented as sets of predicates, and that capabilities can be extracted from the means for achieving goals (e.g., by letting an agent's capabilities be the set of triggering events that match a goal as can be done for AgentSpeak-like languages).

The requirements stem from the fact that the AORTA-component requires access to the agent's mental state in order to perform organizational reasoning that can take the agent's beliefs and goals into account. As noted, we do so by keeping a copy of the mental state, since this allows us to perform reasoning using a dedicated reasoning engine inside the component, rather than relying on the APL. Naturally, keeping a copy of the mental state requires us to perform synchronization in a number of situations:

- **Perceiving the environment:** The cognitive agent is able to perceive changes in the environment using its sensors. Such changes are often relevant for organizational matters (e.g., activation of a norm) and should be made available to the AORTA-component.
- **Performing internal actions:** It is often possible (and useful) for an agent to perform internal actions (i.e., actions that change the internals of the agent, but cannot be seen by a spectator). This could be, e.g., a mental note that states how much the agent wants to spend on a certain item at an auction. This kind of information is surely relevant for the organizational reasoning and it should therefore be made available.
- **Committing to goals (APL):** As we have seen in the AORTA-program for the EAH, the decision to, e.g., commit to an objective can depend on whether the agent has a specific goal. Thus, when the agent commits to achieving a goal, it should be possible for the AORTA-component to use this information.
- **Committing to goals (AORTA):** Naturally, the same applies in reverse: when the AORTA-component executes the commit-action (or drop-action), the agent's goal base should change, since otherwise, the action has no effect.
- **Gaining or loosing capabilities:** In certain APLs, agents are able to gain new capabilities (e.g., in *Jason*, agents can send `tellHow` messages that include a plan



(a) Agent to AORTA. **(b)** AORTA to agent.

Figure 6.6: Flow of information between AORTA and the cognitive agent.

for achieving some goal). Since role enactment in AORTA is heavily based on the notion of capability, it is paramount that the agent's capabilities are available to the component.

- **Communication:** Finally, as we discussed in Chapter 5, we assume that the agents are able to communicate using a messaging system. Since it is possible to send messages using the AORTA-component, we require that the component can "hook up" to the communication system in the agent platform in order to send and receive organizational messages.

Figure 6.6 shows the flow of information in the synchronization process. When the agent commits to a new goal, updates its beliefs, receives a message or gains (or loses) a capability, these changes are propagated via the external agent into AORTA using the Ext-rule. The bridge moves information back to the agent. For example, successful execution of commit(ϕ) will add ϕ to the agent's goal base using the bridge.

Remark (*Predicate translation*). We have chosen Java as implementation language because a large number of existing agent platforms are implemented in Java. Even so, our synchronization mechanism requires a translation of the predicates from the APL to AORTA (and vice versa). AORTA makes use of tuProlog, which means that terms coming from the APL (which are typically implemented using different Java objects based on the type of term: predicate, atom, number, variable, etc.) should be translated into the equivalent objects supported by tuProlog.

6.5 Concluding Remarks

We have presented the AORTA architecture, which consists of an implementation of the metamodel (including a syntax for specifying the model), an implementation of AORTA-programs (including the syntax for the AORTA programming language), and the AORTA-component, which is an implementation of the agent configuration and transition rules specified in Chapter 5.

Our focus on separation of concerns means that the architecture works “out of the box” in the sense that it comes with a complete reasoning engine, but in order to use it with an agent, it should be integrated with the mental attitudes and reasoning process of that agent. We have discussed some of the measures that should be taken in order to do so, including how to synchronize the agent’s mental attitudes with the AORTA mental state, and taking care of translation of predicates during synchronization.

In the next chapter, we present several examples of integrations of AORTA with existing agent platforms.

CHAPTER 7

Programming Organization-Aware Agents

In the previous chapter, we presented the AORTA architecture as an implementation of the operational semantics of AORTA. We have emphasized that our framework is designed as a standalone system that is completely decoupled from any agent platform for three reasons: separation of concerns, ease of integration and verifiability. This chapter focuses on the second point by showing how the architecture can be integrated into existing agent platforms using the connectors between the AORTA-component and the agents in the agent platform.

At this point, it should be clear what we mean by organization-aware agents. We have already discussed what kind of reasoning is required for agents to qualify as organization-aware, and we have shown that the operational semantics of our component (and the architecture) enables that kind of reasoning. However, until now, we have simply assumed the existence of agents with the required capabilities for the roles of an organization and we have completely disregarded the reasoning process of the agents.

In this chapter, we therefore show how to program agents in well-known APLs in which AORTA has been integrated in the agents' reasoning cycle.

The chapter is based on previous work [Jensen, 2015; Jensen et al., 2014], and shows our integration of AORTA with several APLs. In Section 7.1, we present our integration with *Jason*, which is the most complete of our integrations including an integration with the *Jason* IDE. We discuss an integration with 2APL in Section 7.2, and finally, we discuss our integration with the Agent Infrastructure Layer in Section 7.3.

7.1 The *Jason* Agent Platform

The *Jason* agent platform [Bordini et al., 2007] is a Java-based interpreter for an extended version of AgentSpeak. *Jason* is based on the BDI model, is open source, highly extensible and implemented in Java, and is thus ideal as a platform in which AORTA can be integrated. We briefly explain the main concepts of *Jason* before describing how our integration works. Furthermore, to make development of organization-aware agents in *Jason* easier, we have modified to *Jason* IDE to include an editor (with syntax highlighting) for writing AORTA-programs. We end the section with an example of organization-aware agents implemented in *Jason* and AORTA.

The AgentSpeak language is a Prolog-like logic programming language, which allows the developer to create a plan library for each agent in a system. A plan in AgentSpeak is of the form

```
+triggering_event : context <- body.
```

Table 7.1: A simple *Jason*-agent that can sell items in the electronic auction house.

```
1 +have(Item) : not(want(Item)) <- !sold(Item).
2
3 +!sold(Item) : .my_name(Me) & verified(Me) <- !auction(Me,Item).
4 -!sold(Item) : true <- .wait(1000); !sold(Item).
5
6 ...
```

If an event matches a triggering event, the context is matched with the current state of the agent. If the context matches the current state, the body is executed; otherwise, the system continues to match contexts of other plans with the same triggering event. If no plan is applicable, the event fails. Triggering events can be addition or deletion of beliefs (`+l` and `-l`) and addition or deletion of goals (`+!g` and `-!g`). The body contains a sequence of actions the agent should perform and goals to adopt. When adopting a goal in the body of a plan, the agent will attempt to achieve the new goal before continuing executing the current plan.

A part of a simple agent for the electronic auction house is shown in Table 7.1. The example shows three plans: the first plan is triggered when the agent perceives that it has an item (`+have(Item)`), the second is triggered when the agent has an intention to sell an item (`+!sold(Item)`), and finally, the third is triggered if the intention to sell an item fails.

The first plan is triggered by the addition of a belief. The body is executed if the context matches the current state, which it does if the agent does not believe that it wants to keep the item. If that is the case, the body, consisting of a single action, is executed. The action, `!sold(Item)`, adopts a goal to sell the item. This triggers an event, which matches the second plan.

The second plan is applicable if the *internal action*[1] `.my_name(Me)` succeeds with `Me` unified with the agent's name, and the agent has been verified. In that case, it adopts a goal to start an auction for the item. Otherwise, the event fails, which adds a new triggering event (`-!sold(Item)`), which matches the third plan that simply tries to readopt the goal to sell the item.

As mentioned in Chapter 2, an abstract BDI interpreter was proposed, which made it possible to select intentions based on a deliberation of desires and the agent's state. This was further refined with AgentSpeak, on which *Jason* is based. A *Jason*-agent performs its reasoning in the agent *reasoning cycle*, which consists of 10 main steps, including perceiving the environment, updating the belief base and receiving communication from other agents. A number of the steps in the reasoning cycle can be customized to change the agents' behavior, filter incoming messages, alter the behavior of the belief base, or even enable it to connect to a physical robot. In the remainder of this section, we will show how to integrate AORTA into *Jason* by creating customized *Jason* agents. We do not go

[1] Internal actions allow the user to implement features in an object-oriented way by using Java (e.g., calculating how much to bid on an item).

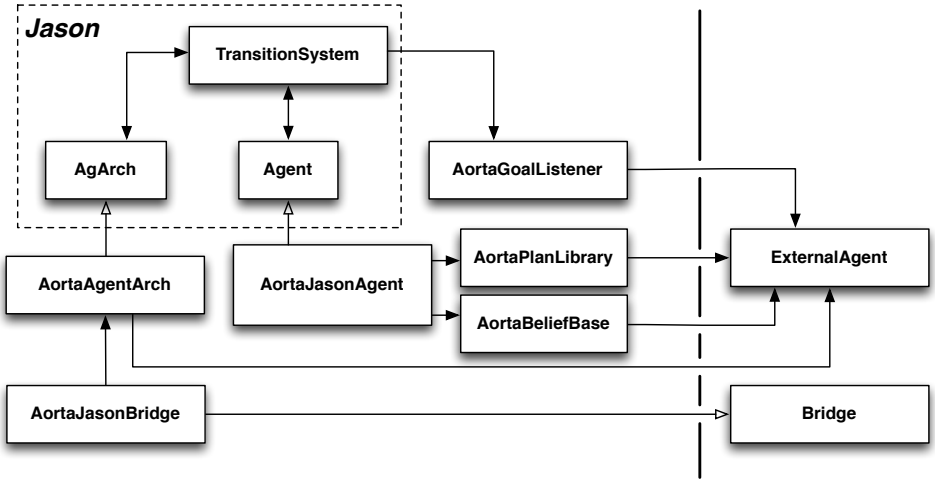


Figure 7.2: Class diagram of the integration of AORTA in the *Jason* agent architecture.

into much detail regarding all the possibilities regarding *Jason*'s extensibility, but refer to [Bordini et al., 2007, ch. 7] for a comprehensive description. We note that the *Jason* extensibility features were adequate for the integration of AORTA.

In AgentSpeak (and therefore in *Jason*), an agent is defined by its beliefs, plans, a number of selection functions (for selecting messages, events, options and intentions), and functions for updating and revising the belief base. All of these can be customized in *Jason* to provide an agent with custom behavior. Furthermore, the agent has an *agent architecture*[2], which is the part of the agent that interfaces with the environment. By customizing the architecture, it is possible to change the agent's perception capabilities, or interface its actions with actuators on a robot. Furthermore, communication with other agents happens through the architecture.

7.1.1 Integration

The AORTA integration in *Jason* is shown in Figure 7.2. The reasoning cycle of a *Jason*-agent is handled by the `AgArch`. Our extension of the agent architecture extends the reasoning cycle with the AORTA reasoning strategy in the beginning of each cycle.

AORTA does not make any changes to the *Jason* language, and any existing implementations of multi-agent systems in *Jason* should be compatible with our integration of AORTA. The integration does a number of things:

- **Beliefs and goals (AORTA):** When transition rules add (or remove) a belief or a goal in the AORTA-component, these changes are translated to *Jason* predicates

[2]Not to be confused with the BDI architecture, which is the agent's reasoning module.

and are propagated to the *Jason*-agent. This is done via `AortaJasonBridge`, which furthermore triggers an addition (or deletion) event, so the agent can react properly.

- **Beliefs and goals (*Jason*):** When the *Jason*-agent is perceiving the environment, adopts a new goal, or performs other actions that change the mental state, AORTA will receive those changes via the `ExternalAgent` and the Ext transition rule. The changes are translated to AORTA-predicates before being added[3].
- **Capabilities:** We define a *Jason*-agent's capabilities as the set of goals `!g` that are triggering events of at least one plan (i.e., `+!g`). Since the plan library can change, we extend it to propagate changes to the agent's capabilities to the `ExternalAgent`.
- **Incoming messages:** Messages from other agents are handled in the `AgArch`. The `AortaAgentArch` processes each message and forwards organizational messages sent by another agent's component to this agent's component via the `ExternalAgent`.
- **Outgoing messages:** The AORTA send-action sends messages via the *Jason*-agent's architecture by forwarding them through the `AortaJasonBridge`.

The system can then be used in a *Jason* project by specifying the classes that each agent should use:

```
1  MAS aorta_project {
2    infrastructure: Centralised
3
4    agents:
5      bob [aorta="reasoning.aorta",model="metamodel.mm"]
6        beliefBaseClass aorta.jason.AortaBeliefBase
7        agentClass aorta.jason.AortaJasonAgent
8        agentArchClass aorta.jason.AortaAgentArch;
9    ...
10 }
```

To make it easier to use, we have implemented an AORTA *infrastructure* for *Jason*, which makes it possible to create an AORTA-project in *Jason* that loads the metamodel, automatically ensures the correct classes are used and that the libraries required for the execution are present (e.g., the tuProlog library). The infrastructure is specified in the *Jason* project file as follows:

```
1  MAS aorta_project {
2      infrastructure: AORTA(organization("metamodel.mm"))
3
4      agents:
5      bob buyer.asl [aorta="reasoning.aorta"];
6    ...
7  }
```

[3]Note that while *Jason* supports annotations on literals (e.g., denoting the source of a belief using an annotation: `verified(carol)[source(mike)]`), they are lost in translation to AORTA formulas, since they are not supported.

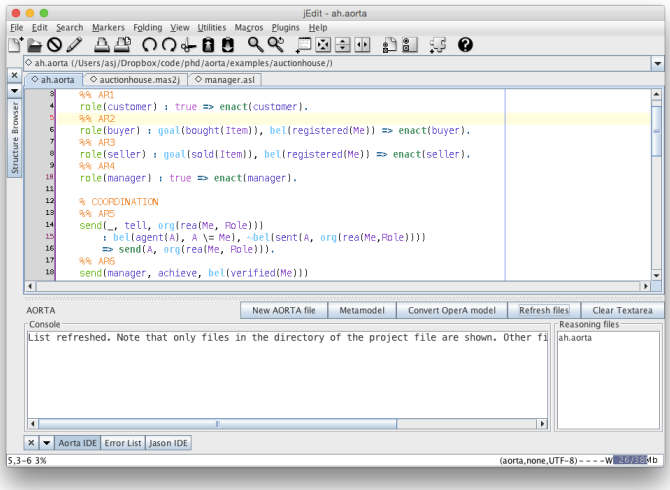


Figure 7.3: The AORTA IDE as part of the *Jason* IDE.

Remark (*Goal semantics*). The semantics of a goal in AORTA is based on the notion of achievement [Hindriks, 2009]: **goal**(ϕ) denotes a goal to *achieve* ϕ, which means that it has been completed once **bel**(ϕ) is the case. This adds flexibility in handling plan failure [Thangarajah et al., 2003], since if the execution of a plan does not achieve the corresponding goal, the goal persists and can be used to select a different plan.

Goals in AORTA are added as intentions in *Jason*. However, an intention for a goal is finished once the plan for that goal has been completed. This means that the goal will be removed from the agent's mental state even though the goal may not have been achieved. However, the AORTA-component is able to commit to the goal multiple times, until it believes it has been achieved.

7.1.2 The AORTA IDE

Jason comes with an integrated development environment (IDE), which enables the programmer to edit the source code using syntax highlighting, continuous parsing providing instant syntax error reports, create agent files and add them to the MAS project automatically, automate the build process, and inspect and debug the agents during execution. The IDE is implemented in jEdit[4], an advanced and highly extensible text editor with numerous plugins available.

The AORTA IDE (see Figure 7.3) is a plugin for the *Jason* IDE, which adds a few new tools and features for programming organization-aware agents: syntax highlighting

[4]http://www.jedit.org/

Table 7.4: Part of the *Jason* program for registering and putting items up for sale.

```
1  have("PH-lamp").
2  +have(Item) <- !sold(Item).
3
4  +!registered(_) <- register("Address", "Account").
5
6  +!sold(Item) : .my_name(Me) & verified(Me) <- !auction(Me,Item).
7  +!sold(Item) <- .wait(1000); !sold(Item).
8
9  +!auction(Me,Item)
10   : .my_name(Me) & registered(Me,_,_) &
11     have(Item) & not(auction(_,Item,Me,_,_))
12   <- start_auction(Item, 10, 5).
13 +!auction(Me,Item) <- .wait(1000); !auction(Me,Item).
```

Table 7.5: Part of the *Jason* program for verifying the credentials of participating agents.

```
1  +!verified(Agent)
2    : not(badInfo(Agent)) & registered(Agent,_,_)
3    <- verify(Agent).
```

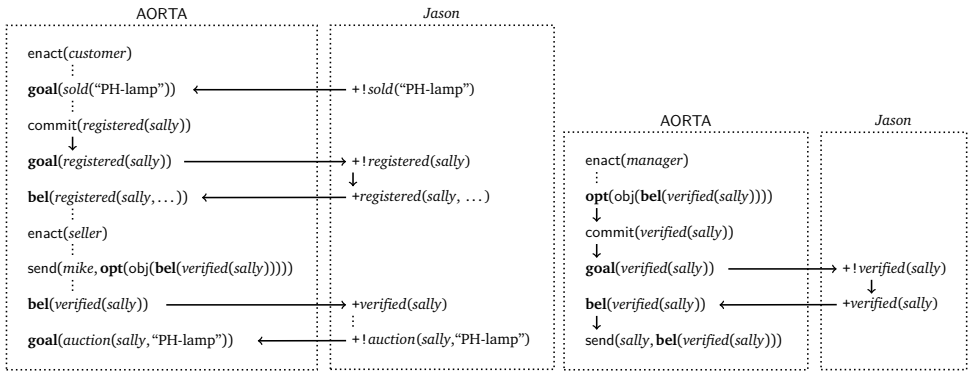
of AORTA-programs and the metamodel, continuous parsing and syntax error reporting, and conversion of an OperA model to the AORTA metamodel (see Appendix B). These features make using AORTA together with *Jason* more pleasant, since errors are quickly detected, and the (*Jason*) agents can be executed to test the implementation immediately.

7.1.3 The electronic auction house scenario

We show how to implement *Jason*-agents that can use AORTA to participate in the electronic auction house. Our focus is on the interaction between the agent's cognitive reasoning and the organizational reasoning of AORTA, so our implementation does not include bidding strategies and other advanced techniques that would be expected of auction-agents.

We show the execution of the registration and verification process for Sally, who wants to sell a PH-lamp. The program for Sally is shown in Table 7.4 and the program for Mike (handling the verification) is shown in Table 7.5. We refer to Appendix A.3 for the full implementation of the *Jason* agents.

Figure 7.6 shows how AORTA interacts with *Jason* during an execution. For the step-by-step execution of the AORTA-component, we refer to Figure 5.5 in Chapter 5. In Figure 7.6(a), *Sally* adopts the goal of selling her lamp (line 2 in Table 7.4). This is added to the AORTA mental state via the `ExternalAgent`. This activates the rule to commit to the registration-objective and she eventually executes it. The newly adopted goal is added to *Jason* via the bridge, and the agent completes the objective using a plan in her



(a) Sally **(b)** Mike

Figure 7.6: The flow of execution starting when Sally enacts the customer role, and Mike enacts the manager role. Dots indicate that the following event is not directly related to the previous, but that it happens afterwards (e.g., Mike obtains the verification-option after executing enact(*manager*), but not as result of the execution).

plan library. The belief is transferred to AORTA and she will execute the rule to enact the seller-role. She will send her verification-objective to Mike, who commits to verifying her (see Figure 7.6(b)). His goal is added to *Jason*, and he will execute a plan to verify her. Finally, he will inform her about the verification, which leads to her creating an auction for the lamp.

Note that while it may seem like the agent is compelled to commit to the objectives given the organizational rules, it is important to emphasize two points: first, in other cases there will be more rules to choose between; rules that may influence the agent in different directions.

Second, the agent can deliberately choose *not* to complete an objective. In *Jason*, the *intention selection function* can be changed to, e.g., prioritize the agent's own goals (see Part IV). This, of course, is only valid in the current integration; other frameworks may not have this possibility. However, the agent can deliberately skip parts of an objective, or even the entire objective, simply by marking the objective as done (i.e., a belief addition in *Jason*). This is a deliberate violation of the expectations of the role, but nothing prevents the agent from doing so. Since other agents would perceive that the objective was *not* completed, they would be able to detect a violation and choose to sanction the agent.

For example, if Bob is a good friend of Mike, it is possible for Mike to circumvent the organizational prohibition of *not* verifying Bob using the following program:

```
1 +registered(bob) : true <- verify(bob).
```

Of course, other agents are able to conclude that Mike has violated the prohibition and may choose to sanction Mike (and Bob) if they are able to.

7.2 2APL: A Practical Agent Programming Language

2APL (or A Practical Agent Programming Language) is a BDI-based agent-oriented programming language that "*realizes an effective integration of declarative and imperative style programming by introducing and integrating declarative beliefs and goals with events and plans. It also provides practical programming constructs to allow the generation, repair, and (different modes of) execution of plans based on beliefs, goals, and events*" [Dastani, 2008]. As with many other agent platforms, 2APL is implemented in Java (based on a formal syntax and semantics) and has been used for various research purposes [Alechina et al., 2012; Dybalova et al., 2014; Westra, 2011].

A 2APL agent is composed of a number of *ingredients* that specify various aspects of the agent. These include beliefs, goals, belief update actions (similar to STRIPS actions), external actions (i.e., actions in an environment), plans, and different kinds of practical reasoning rules.

Three types of practical reasoning rules are proposed in 2APL: planning goal rules (PG), procedure call rules (PC) and plan repair rules (PR). PG-rules are used to generate plans when the agent has certain goals and beliefs. PC-rules are used to respond to messages and external events. PR-rules can be used to repair a failed plan by revising parts of the plan. Each rule is of the form:

```
head <- condition | body.
```

Exactly what can be specified in each part of the rule depends on the type of rule. We refer to [Dastani, 2008] for details.

The 2APL deliberation cycle is composed of a number of steps, including applying PG-rules, execution actions, and processing events and messages. The agent then decides whether to perform a new deliberation cycle, which only makes sense if some of the steps actually did "something" (e.g., an action was executed, a message was received or an event was processed). If nothing happened, the agent will sleep until it receives a message or some external event happens.

7.2.1 Integration

Compared to *Jason*, 2APL is not as easily extended to incorporate the AORTA-component. By this, we mean that at some places the implementation is inflexible and requires us to change, rather than extend, classes of 2APL. These changes are, however, not major, and in most cases the incorporation can be done quite smoothly; for example, no changes to the 2APL syntax is required.

The integration of AORTA in 2APL is shown in Figure 7.7. The deliberation cycle of each agent is extended in `AortaAPLModule`[5]. We extend the belief base, goal base and PG-rule base to make synchronization possible. The `AortaMessenger` handles all messages: organizational messages are put into the AORTA-component, while other messages are processed by the standard 2APL messenger.

[5] In the 2APL implementation, a *module* refers to an agent.

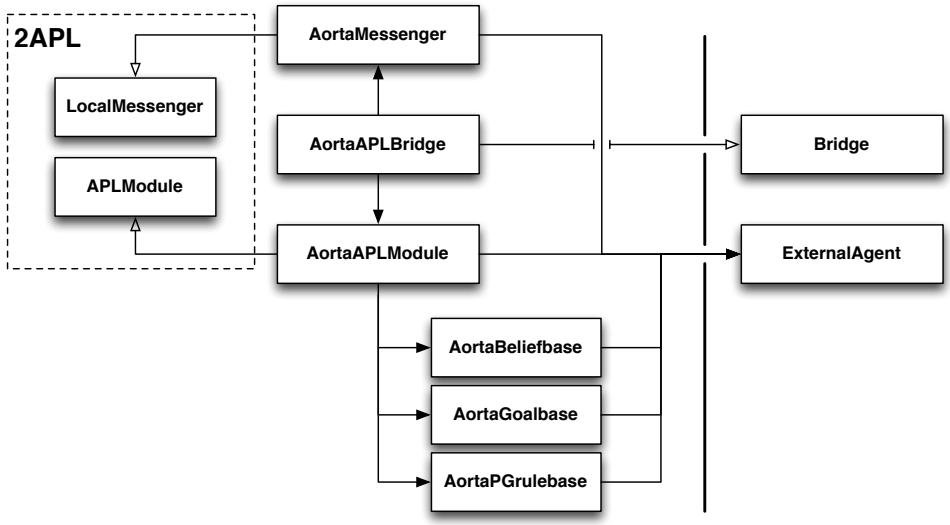


Figure 7.7: Class diagram of the integration of AORTA in the *Jason* agent architecture.

A few remarks about the implementation are warranted:

- It is possible in 2APL to specify *conjunctive goals*, e.g., $x \wedge y$, which is only achieved in a state where *both* x and y are true. This is in contrast with a set of goals x and y, which can be achieved by independently achieving each. Unfortunately, AORTA does not support this kind of goal and they can as such not be used.

  We can, however, exploit the fact that the belief base of a 2APL agent is a Prolog program, and define a rule:

  ```
  z :- x, y.
  ```

  Then, by adopting the goal z, the agent is committed to achieving $x \wedge y$. Note that this makes reasoning about such conjunctive goals harder, but it provides a simple way to circumvent the limitation.

- We define a 2APL agent's capabilities as the set of goals that are specified as head in *PG-rules*. We do so, because these rules are the only ones that specifically target goals, and our notion of capability is based on the ability to achieve goals.

- The 2APL deliberation cycle puts the agent to sleep if nothing has happened in a single cycle. Our integration needs to handle this, since the AORTA-component might alter the agent's state. Therefore, the deliberation cycle is changed as follows. First, the AORTA organizational cycle is executed. If the AORTA configuration changed during the execution, we add a flag in the deliberation cycle. Second, the normal deliberation cycle is executed, and when it checks whether the configuration was

Table 7.8: Part of the 2APL program for registering and putting items up for sale.

```
beliefs:
  have(lamp).

pgrules:
  <- have(X) | { adopta(sold(X)); }
  registered(Me) <- true | { @ah(register(x,y), S); }
  sold(Item) <- me(Me) and verified(Me) | { adopta(auction(Me,Item)); }
  auction(_, Item)
    <- me(Me) and registered(Me,_,_) and have(Item) and
      not auction(_,Item,Me,_,_) |
    { @ah(startAuction(Item, 5, 10), S); }
```

Table 7.9: Part of the 2APL program for verifying the credentials of participating agents.

```
pgrules:
  verified(Agent) <- registered(Agent) and not badInfo(Agent) |
    { @ah(verify(Agent), S); B(S = ok); +verified(Agent); }
```

changed, it checks the flag added by AORTA as well. Thus, the agent executes another cycle if the AORTA-component altered the state, even if the deliberation cycle did not.

7.2.2 The electronic auction house scenario

We have implemented the electronic auction house scenario for the 2APL agents as well. The execution of the agents is similar to that of the *Jason* agents (from our perspective), since the flow of information between AORTA and 2APL is the same.

The most interesting fact is that to implement the scenario, we only had to implement the 2APL agents; the metamodel and AORTA-program can be reused as-is. For example, consider the parts of the 2APL program for registering, verifying and putting items up for sale shown in Table 7.8 and Table 7.9.

Since the 2APL agents have the capability to register, the AORTA-component lets them enact the customer role and subsequently perform the registration process. All of this was possible using the existing AORTA-program and the few lines of code shown in the table.

We refer to Appendix A.4 for the full implementation of the 2APL agents.

7.3 AIL: The Agent Infrastructure Layer

As the final example of an integration of AORTA with an existing agent platform, we present our integration with the *Agent Infrastructure Layer* (AIL). AIL is not an APL as such [Dennis et al., 2012, p. 10]:

> *The Agent Infrastructure Layer is an intermediate layer that encompasses key concepts from a wide range of BDI programming languages as data structures in Java and enables the implementation of their operational semantics within a clear framework.*

In other words, AIL enables us to implement a wide range of BDI-enabled APLs as long as they are described by operational semantics. As previously mentioned, a variety of APLs with operational semantics have been proposed, including *Jason*, GOAL and 2APL. These (and other) APLs can with relative ease be implemented in AIL (e.g, it is shown in [Dennis et al., 2012] how to implement the operational semantics of GOAL). Furthermore, AIL is interfaced with an underlying *agent model checker* enabling verification of agent systems.

By integrating AORTA with AIL, we accomplish two things:

- AORTA can be integrated with any BDI-enabled APL implemented in AIL.
- We can use model checking to verify systems that use AORTA (see Chapter 8).

We dedicate Chapter 8 to verification of AORTA, but since the results depend on our integration choices in this chapter, we briefly discuss the different approaches we can take. The fact that AIL is tailored for the implementation of operational semantics of APLs means that we can implement the AORTA operational semantics in AIL. This provides a more tight integration with the APL (and more control over the verification process), but it comes with a few downsides when considering verification. Since it is another implementation of the operational semantics, we have to convince ourselves that the implementation is equivalent to that of the AORTA architecture, for the verification to be useful for practical purposes[6].

On the other hand, since AIL uses Java, it is relatively straightforward to integrate AORTA similarly to the integration with *Jason* and 2APL. The verification is thus more convincing, since the actual system is verified, but since the system was not implemented for model checking, it will not benefit from the optimizations provided by AIL, and the process might take very long time in comparison with an implementation of the operational semantics.

In the end, we implemented both approaches: first, we provided a standard integration similar to those of *Jason* and 2APL, but for model checking purposes, this was not adequate. For example, to verify that *registered*(*carol*) eventually holds in the system took 18 hours! In order to optimize this, we therefore implemented the operational semantics and the organizational cycle in AIL to benefit from its efficiency improvements. In the following, we describe the implementation of the operational semantics in AIL, and in Chapter 8, we present verification results of both implementations.

7.3.1 Implementation

AIL enables the implementation of BDI-based APLs with operational semantics, so naturally, we want our implementation to work with any such APL. The implementation

[6]This, of course, is not an issue pertaining only to AORTA but to any APL that is implemented in AIL

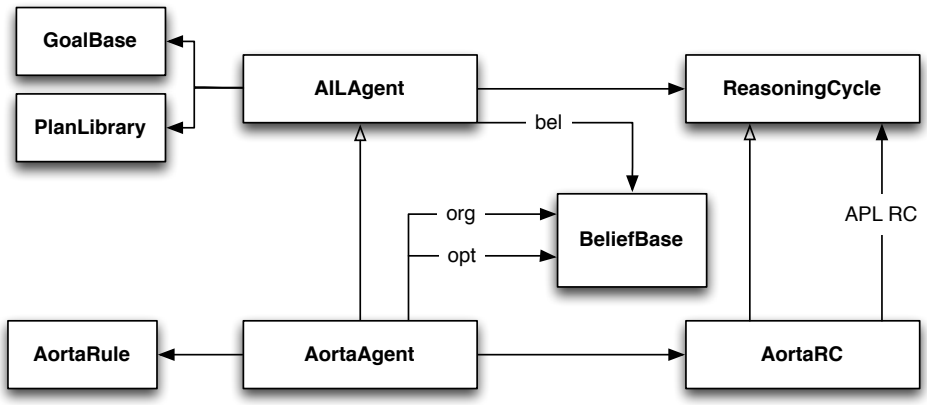


Figure 7.10: Class diagram of the implementation of the AORTA operational semantics in AIL.

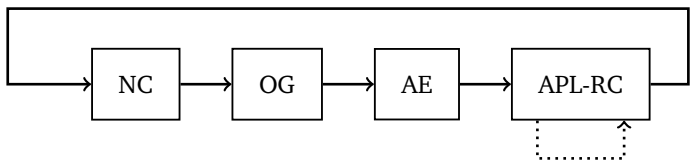


Figure 7.11: The combined reasoning cycle of AORTA and the APL agent in AIL.

of an APL agent in AIL is done by extending the `AILAgent` class. This class contains the belief bases, goal bases, plan library, reasoning cycle, and intentions of an agent. In many cases, it will be sufficient to implement a new reasoning cycle (by implementing the `ReasoningCycle` interface), but it is also possible to override methods for, e.g., selecting an applicable plan.

In our implementation, which is shown in Figure 7.10, we first build the APL agent, and then feed it into the `AortaAgent`, which extends `AILAgent`. Here, we copy all data structures from the APL agent into the AORTA-agent, which thus comprises the APL agent and the AORTA-component. However, if the APL agent overrides methods in `AILAgent`, these are lost, so in that case, we need to perform a number of manual changes to the `AortaAgent`, but for our purpose, this is acceptable.

The `AILAgent` contains a belief base, which contains the agent's beliefs, but it is designed such that other belief bases can be added. We thus add belief bases for organizational beliefs and options, and can immediately query them using AIL. Furthermore, we add a number of `AortaRules`, which represent the AORTA reasoning rules. For this purpose, we have chosen not to implement if-rules, since they are syntactic sugar and can easily be transformed to reasoning rules.

Table 7.12: Gwendolen plans for registering and putting items up for sale.

```
1 +!registered(Me) [achieve] : { ~B registered(Me) }
2   <- register(address, account), +!registered(Me) [achieve];
3
4 +!sold(Item) [achieve] : { B me(Me) }
5   <- *verified(Me), +!auction_created(Me,Item) [achieve];
6
7 +!auction_created(Me,Item) [achieve] : { ~B auction(Me,Item) }
8   <- start_auction(Item, 10, 1), *auction(Id,Item,Me,SP,ET),
9     *auction_done(Id,Winner,Bid), +sold(Item);
```

The `AortaAgent` implements an `AortaReasoningCycle`, which contains a reference to the APL agent's reasoning cycle and works as shown in Figure 7.11. First, we note the initial stage of the APL agent's reasoning cycle. Second, the organizational cycle is executed. Third, we execute the APL agent's reasoning cycle until the initial stage is reached. We then repeat the cycle. By doing so, we effectively integrate the AORTA reasoning cycle with the APL agent's reasoning cycle.

7.3.2 The electronic auction house scenario

Following the previous integrations of AORTA, we have implemented the electronic auction house scenario to be executed by AIL-enabled agents as well. The current version of AIL includes only a proof of concept language called Gwendolen [Dennis and Farwer, 2008] (though both GOAL and SAAPL are referred to as example implementations [Dennis et al., 2012]). The Gwendolen language was developed by Dennis et al. during the work on AIL and it provides the default semantics for AIL.

The syntax of Gwendolen is somewhat similar to that of *Jason*, in that the agents' functionality is based on plans of the form

```
event : guard <- deed*
```

where events may be perception, commitment to goals, and incoming messages. A deed is a statement that can appear in the body of a plan and can be an action, a belief addition or deletion, and a goal addition or deletion. It represents things the agent is planning to do, but has not done yet. Finally, a guard is a formula that must be believed by the agent before the plan is applicable.

Table 7.12 lists the plans for registering and putting items up for sale in the auction house. An asterisk in front of a belief b, `*b`, denotes a *wait for*-action that suspends the intention until the agent believes b. For example, the plan to sell an item consists of first waiting to become verified and then to adopt the goal to create the auction. The verification process is handled by AORTA by delegating the verification objective to the manager.

7.4 Concluding Remarks

This chapter presented three integrations of AORTA into APLs: first, a rather large integration with *Jason* including an integrated development environment, syntax highlighting, and instant syntax error detection. Second, a proof of concept integration with 2APL, which shows that the component is in fact truly decoupled from specific APLs, and third, an implementation of the AORTA operational semantics in AIL to enable 1) integration with potentially any BDI-based APL based on operational semantics and 2) verification of organization-aware agents.

In Chapter 8, we focus on the second point of the AIL integration, and show how our implementation of the operational semantics of AORTA can be used to verify programs executed by organization-aware agents.

CHAPTER 8

Model Checking AORTA

In many of the areas where MASs are used, there is a need for dependability and security (e.g., air traffic control [Ljunberg and Lucas, 1992], autonomous spacecraft control [Muscettola et al., 1998] and health care [Moreno and Garbay, 2003]). Therefore, it is increasingly necessary to consider formal verification of such systems [Bordini et al., 2008]. As organization-oriented MASs become more popular, we will see a need to verify organizational properties of agent systems as well. In some ways, organization-aware agents will naturally tend to be more complex than their "unaware" counterparts; even though programming them may be easier since certain aspects may be automated (task allocation, coordination, etc.), the reasoning cycle of the agents will include more steps. This can make it even harder to convince ourselves that our implementation is correct.

Though AOP is a specialization of OOP, the agent metaphor is radically different from the object metaphor, and in order to capture this, the verification techniques from OOP must be extended. That is, since an agent is autonomous and its behavior is based on beliefs and intentions, we need to be able to not only check *what* the agent has done (similar to verification in OOP), but also *why* it did so (e.g., what did it believe and what intentions did it have). In relation to organizations and the AORTA framework, we need to be able to verify a number of things:

1. The agents can use the organizational structure (e.g., enact roles).
2. If violation of norms occur, the system still functions properly.
3. The proper organizational options are generated to be acted upon.

Baier and Katoen [2008] define the principles of model checking as "*an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model*". Model checking APLs thus means translating the system into a finite-state model in which we can prove certain properties. Since agents are usually enriched with mental attitudes such as beliefs, goals and intentions, model checking APLs is only interesting, if we can verify properties about these mental attitudes. For example, it should be possible to check whether agents in a system only intend to achieve goal states as described by the temporal formula $\Box(\mathbf{I}(ag, \phi) \rightarrow \mathbf{G}(ag, \phi))$. Here, ag refers to an agent, ϕ is a state and $\mathbf{I}$ and $\mathbf{G}$ are modal operators referring to intentions and goals, respectively. Quite some work has been done to make it possible to perform model checking on existing APLs, and most notably, the Agent Java PathFinder (AJPF) project [Dennis et al., 2012] is an example of a practical system in which model checking can be feasible.

In this chapter, we present an extension to AJPF, which makes it possible to perform verification of organization-aware agents using AORTA. We extend the specification language of AJPF to incorporate modalities about organizational information.

The chapter is organized as follows. In Section 8.1, we describe how to specify properties concerning mental attitudes of agents and how the AJPF model checker works. We discuss more recent work on how to model check multi-agent systems that are regulated by an organization and how verification of systems using AORTA can be done in Section 8.2. Finally, we use model checking to verify various properties of (a simplified version of) the EAH scenario in Section 8.3.

8.1 Model Checking Agent Programming Languages

Much of the work done in the area of model checking MASs and agent programming languages has been in the setting of AgentSpeak(L) [Bordini et al., 2004a; Bordini et al., 2006; Bordini et al., 2004b]. While interesting, such approaches are generally hard to extend to other languages without a lot of hard work. Furthermore, in such setting, verification of heterogeneous MASs[1] is not possible. Recently, Doan et al. [2014] have proposed a way to verify heterogeneous MASs by translating programs into a common metalanguage, meta-APL. However, since this approach requires a translation of the program, we need to convince ourselves that the translation is faithful to the original program.

In this chapter, we focus on another approach for verifying agent systems, which is based on an extended version of Java PathFinder (JPF) [Visser et al., 2003] called Agent JPF (AJPF), which takes advantage of the advanced model checking features of JPF, while making it possible to verify properties relevant to intelligent agents. AJPF can be used as-is for potentially any APL implemented in Java, but its real power shows, when combined with the *agent infrastructure layer* (AIL), which was mentioned in Chapter 7. AIL has been optimized for model checking in AJPF by making use of a JPF feature for matching states (to avoid performing redundant checks) and by means of a *choice generator* that ensures all (meaningful) interleavings of agents are investigated (see Section 8.1.2).

8.1.1 Specifying Properties

In this section, we describe the temporal logic used for specifying properties relevant for multi-agent systems – the *property specification language* (PSL) [Dennis et al., 2012]. We use this language to be able to specify high level agent concepts (in PSL) to be verified at a lower level (in AJPF, where verification is done at Java bytecode level).

The temporal logic usually used for model checking multi-agent systems is linear-time temporal logic (LTL) with the addition of modal operators for beliefs, goals, intentions, actions and percepts [Bordini et al., 2008; Bordini et al., 2006; Dennis et al., 2012]. We can thus use PSL to specify formulas that should hold in a given system. For example, we might want to verify that an agent *ag* only intends to achieve states ϕ that it does not

[1] That is, MASs comprised of agents implemented in different APLs.

believe have already been achieved:

$$\Box(\mathbf{I}(ag, \phi) \rightarrow \neg\mathbf{B}(ag, \phi))$$

The full PSL syntax is given below, where ag is the agent's name, and f is a ground first-order atomic formula.

$$\phi ::= \mathbf{B}(ag, f) \mid \mathbf{G}(ag, f) \mid \mathbf{D}(ag, f) \mid \mathbf{I}(ag, f) \mid \mathbf{P}(f) \mid \phi \vee \phi \mid \neg\phi \mid \phi \mathsf{U} \phi \mid \phi \mathsf{R} \phi$$

$\mathbf{B}(ag, f)$ is true if agent ag believes f to be true, $\mathbf{G}(ag, f)$ is true if the agent has f as a goal. **D** represents actions (*done*), **I** intentions and **P** properties of the environment. The LTL formulas U and R represents "until" and "release", respectively. The temporal operators $\Diamond$ (eventually) and $\Box$ (always) can be derived from U and R.

The underlying semantics of the modal operators depends on the multi-agent system being verified. We can consider a generic multi-agent system (which could be an AIL-enabled MAS, but also any other APL implemented in Java), where *MAS* is the state of the system at a point of time in the run. Furthermore, let $ag \in MAS$ be an agent in the program execution at this point. The semantics of beliefs is then

$$MAS \models \mathbf{B}(ag, f) \quad \textit{iff} \quad ag \models f.$$

Here, $\models$ is logical consequence as implemented by the specific agent programming language. The semantics of the other modal operators can then be given in a similar way.

8.1.2 Agent Java PathFinder

Agent JPF is a module for the Java PathFinder. JPF consists of an implementation of the Java Virtual Machine (JVM), which can execute all paths through a program in order to verify some predefined properties about the program. Since the state space is often huge, JPF employs state matching in order to reduce the number of states explored.

AJPF implements a controller, which takes care of execution of each of the agents in the system. At each time step, it checks whether the system is in an end state[2] and should terminate, and otherwise it decides which agent to execute. This decision is made by a *scheduler*, which keeps a list of *active* agents, i.e., agents that have more work to do (i.e., in the form of active intentions). The model checker can then branch out at each of these states and execute one of the active agents (by choosing one path, executing it until reaching an end state and then backtracking), as illustrated in Figure 8.1.

The design of the scheduler influences the efficiency of the model checker. The default scheduler of AJPF is a simple scheduler, which chooses between every active agent at each state. This means that all interleavings of agent execution will be checked, making it likely to generate a lot of unnecessary states. AJPF comes with another scheduler, the *action scheduler*, which chooses an agent and executes *only* this agent until something changes in the system (new percepts, the agent goes to sleep, etc). This will effectively

[2] An end state is defined as a state where all agents are sleeping and the environment is not changing, thus nothing new can happen after such state has been reached.

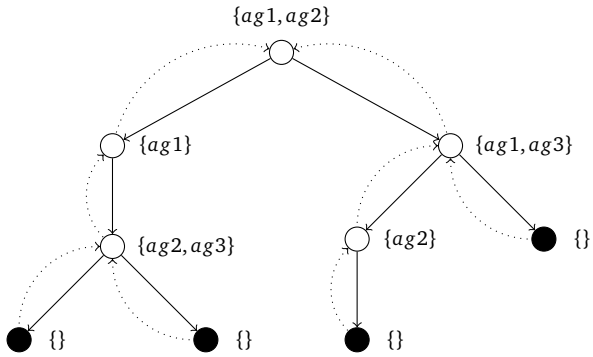


Figure 8.1: Branching and backtracking in AJPF. Each state has a set of active agents. The filled circles denote end states and the dotted edges indicates that AJPF backtracks.

cut off these interleavings, when they are unnecessary, thus speeding up the verification. If some parts of the system requires to be continuously operational, then this will not work, since these parts of the system will only be updated when the current agent takes action.

Each agent in an APL (either an AIL-enabled or an existing APL) needs to implement the `MCAPLLanguageAgent` interface, which is used by AJPF to perform all the steps necessary for the verification of a system:

- Perform a reasoning step (`MCAPLreason()`).
- Put the agent to sleep (`MCAPLwantstosleep()`) or wake it up (`MCAPLwakeup()`).
- Check if a property holds (`MCAPLbelieves(fml)`, `MCAPLhasGoal(fml)`, etc.).

AJPF provides a listener, which is used to verify the properties specified in PSL. This is done by first building a Büchi automaton [Sistla et al., 1987] that represents the negation of the property, and then compute the global behavior of the system by executing it. The product of these, the product automaton, can then be used to check if the property is violated. A property is violated if there exists a path to an accepting state [Courcoubetis et al., 1993]. The implementation of the model checker in AJPF employs techniques that allows progressively building the product automaton [Gerth et al., 1995], making the verification process more efficient, since properties can be checked on-the-fly rather than after the system has finished executing[3].

[3]Furthermore, some systems might be designed without an end state (e.g., a surveillance agent that continuously surveys an area), in which case it is necessary to be able to check properties on-the-fly (or employ other techniques, such as specifying a max depth of a branch in the model).

8.2 Verification of Organizations in Multi-Agent Systems

Dignum and Dignum [2012] define several desirable properties of organizational MASs: what makes an organization e.g., well-defined, good or efficient. These properties require not only a way to express the beliefs of the agents in the system, but also the state of the organization. For example, a good organization *"is an organization such that if the organization has the capability to achieve ϕ and there is a group of roles in the organization responsible for realizing it, then the roles being in charge have a chain of delegation to roles that are played by agents that are actually capable of achieving it"* [Dignum and Dignum, 2012, p. 302]. Being able to verify that a system satisfies these properties would be a large step towards convincing oneself that the system actually works. Verification of organizational aspects has been investigated before [Astefanoaei et al., 2008; Dennis et al., 2010; Huget et al., 2002; Viganò, 2007], but usually only by considering the internals of each agent as a black box. Model checking of electronic institutions specified in the ISLANDER framework was explored by Huget et al. [2002]. By translating an ISLANDER specification into MABLE, a language for automatic verification of MASs, the system can be verified using the SPIN model checker.

The work most similar to ours is described in [Dennis et al., 2010], where a programming language for normative MASs is implemented in AIL. Agents in the system can interact with an organization, and the system can then verify various properties of both the agents and the organization. Our integration with AIL differs in that we only verify properties of the agents, but these properties may include organizational properties, as defined in the AORTA-component. Since AORTA is not tightly coupled to a specific APL, the integration with AIL allows us to perform verification of existing agents with additional properties concerning an organization, and not just agents programmed in a specific normative language.

8.2.1 Specifying organizational properties

In order to verify properties about organizational beliefs and options, we need to be able to express the properties in PSL. We therefore extend the PSL syntax to incorporate such properties:

$$\psi ::= \phi \mid \mathbf{Org}(ag, f) \mid \mathbf{Opt}(ag, f)$$

The interpretation of $\mathbf{Org}(ag, f)$ is given as:

$$MAS \models \mathbf{Org}(ag, f) \quad \text{iff} \quad MS_{ag} \models \mathbf{org}(f),$$

where *MAS* is the multi-agent system (AIL+AORTA) and MS_{ag} is agent ag's mental state as defined in Chapter 5. Similarly, the interpretation of $\mathbf{Opt}(ag, f)$ is:

$$MAS \models \mathbf{Opt}(ag, f) \quad \text{iff} \quad MS_{ag} \models \mathbf{opt}(f).$$

In AJPF, we have implemented the extended PSL by adding functions that check whether an agent has organizational beliefs or options to the `MCAPLLanguageAgent` interface, which defines the methods needed by AJPF to perform model checking. We then

make it possible to verify organizational properties in AIL agents by implementing the methods in the `AortaAgent` class of our extension to AIL:

- `hasOrganizationalBelief(Formula phi)`: Returns true if Σ_o contains `phi`.
- `hasOrganizationalOption(Formula phi)`: Returns true if Γ_o contains `phi`.

The AJPF system can then be used to perform verification of agents with an AORTA-component, since we have provided an implementation of AORTA in AIL, and have added the ability to verify properties about organizational beliefs and organizational options to AJPF.

8.3 Evaluation

We evaluate AORTA using the model checker in three steps: first, we show that various properties of the EAH scenario can be verified. Second, we discuss how the number of agents influence the size of the state space. Third, we show how we can verify certain general, desirable properties of organizations (based on the properties defined in the LAO formalism [Dignum and Dignum, 2012]).

Remark (*Communication through AIL*). The operational semantics of AORTA assume available some messaging mechanism in the underlying APL. In our implementation of the semantics in AIL, we thus used the AIL messaging feature for communication in AORTA. This, however, leads to a very large number of states because of the possible ways of interleaving the agents' execution.

When all the agents inform each other by their role enactment (which in our scenario is one of the first things to happen), this generates a very large state space that quickly makes verification of the system infeasible. We thus decided to change the implementation, such that communication between AORTA-components happens directly (i.e., by putting new messages directly into the AORTA-components inbox).

When verifying, e.g., the property $\Diamond\mathbf{B}(\texttt{sally}, \mathit{registered}(\mathit{sally}))$ in the simple version of the scenario, we thus reduced the number of states searched from 756 to 160, and it became feasible to verify other properties, such as norm fulfillment, which requires us to search a larger part of the state space.

Furthermore, the change made it possible to actually verify (parts of) the full scenario, though the time required to verify most of the properties still makes it a tedious exercise.

8.3.1 The electronic auction house scenario

As noted, verification requires us to search through the state space of a program to verify that certain properties will eventually hold (or will never hold). Naturally, the size of the state space depends on the size of the program, and during our investigation, we realized that our implementation of the EAH scenario generates a rather large state space, which for practical purposes is infeasible to perform model checking on. We have thus created a reduced version of the scenario, which ends once an auction has been created. Furthermore, the number of agents was reduced to three (Carol, Mike and Sally). While this

simplifies the system quite much, it is still possible to verify certain interesting properties of the system.

The system was evaluated using the properties listed in Table 8.2. We verified some of the properties in the full version of the scenario as well, but were unable to do so for others. Furthermore, we attempted to verify some properties using an integration of the AORTA architecture with AIL, but as the results show, it quickly became infeasible. We were simply unable to verify most of the properties, so in the following we focus on the verification of the optimized version of AORTA.

We do not go into details with how much *time* was used for verifying each property since this depends on the machine used, but to get an idea of the feasibility of the approach, we note that while property 1 took 1:54 minutes to verify[4] in the simpler version of the auction house scenario, it took 48:16 minutes in the full version of the scenario. From this, it should be clear that the time used quickly makes the process infeasible (or impractical at best).

Properties 1–5 check various properties regarding the agents' beliefs. For example, that Carol will eventually become verified (property 4) and that Sally will eventually have created an auction (property 5). Properties 6–15 show that various organizational properties can be verified. For example that Mike enacts the manager-role (property 10) and that Sally has the option to become registered until she commits to that goal (property 13).

As the results show, as soon as we move from the simple version of the scenario to the full version, the number of states greatly increases. While an increase is expected since we add another agent, the difference in, e.g., property 7 is greater than we would have expected. We suspect that the issue lies within the state matching functionality, such that even though we would consider two states identical, AJPF is unable to match them. Unfortunately, we were not able to pinpoint the exact cause of this.

Verifying parts of a system

Since our approach has some issues with regards to state matching, it is difficult to verify properties that require us to search deeply into the state space. One way to circumvent this is to create an alternative implementation of the program, which begins execution at a later time in the scenario. Of course, by verifying properties in this way, we do not prove that the properties hold in the full program, but it provides a way to convince ourselves that the system works.

We have verified a number of properties regarding Bob's violation of the norms in a setting, where Sally has already created the auction. We limit the system to two agents: Bob and Mike, and Bob has to enact his roles, register and try to bid on the item. The properties are listed in Table 8.2. The results show that in a reduced system like this, it is feasible to verify properties.

For example, we have verified three related properties: 1) Bob will never become verified, 2) Mike will not violate his prohibition and 3) if Bob violates his obligation to

[4]We evaluated the system using Java 7 on a laptop with a dual core 2.80 GHz Intel i7 CPU and 8 GB RAM running Windows 8.1.

Table 8.2: The properties that were verified by AJPF for the simple[1] and full[2] version of the scenario, and the simple scenario in the AORTA architecture implementation[3].

	Property	States[1]	States[2]	States[3]
Agent properties				
1	$\Diamond$**B**(sally, *registered*(*sally*))	160	2548	16765
2	$\Diamond$**B**(carol, *registered*(*carol*))	1027	53926	—
3	$\Diamond$**B**(sally, *verified*(*sally*))	706	14056	—
4	$\Diamond$**B**(carol, *verified*(*carol*))	1229	—	—
5	$\Diamond$**B**(sally, *auction*(*sally*, *lamp*))	1010	—	—
Organization properties				
6	$\Diamond$**Org**(sally, *rea*(*sally*, *customer*))	22	142	66
7	$\Diamond$**Org**(sally, *rea*(*sally*, *seller*))	160	2548	1192
8	$\Diamond$**Org**(carol, *rea*(*carol*, *customer*))	96	1878	212
9	$\Diamond$**Org**(carol, *rea*(*carol*, *buyer*))		53926	
10	$\Diamond$**Org**(mike, *rea*(*mike*, *manager*))	62	658	170
11	$\Diamond$(**Org**(carol, *rea*(*mike*, *manager*)) $\wedge$ **Org**(sally, *rea*(*mike*, *manager*)))	62	658	2256
12	$\Diamond$(**Opt**(sally, *obj*(*verified*(*sally*))) R **Opt**(mike, *obj*(*verified*(*sally*))))	294	4960	—
13	$\Diamond$(**Opt**(sally, *obj*(*registered*(*sally*))) U **G**(sally, *registered*(*sally*)))	74	894	3580
14	$\Diamond$(**B**(sally, *registered*(*sally*)) R **Org**(sally, ϕ))	160	2548	16363
15	$\Diamond$((**B**(sally, *registered*(*sally*)) R **Org**(*sally*, ϕ)) U (**B**(sally, *verified*(*sally*)) $\wedge$ $\neg$**Org**(sally, ϕ)))	842	—	—
Bob's violation properties				
16	$\Diamond$**B**(bob, *registered*(*bob*))	34		
17	$\Box\neg$**B**(bob, *verified*(*bob*))	154		
18	$\Diamond$**B**(bob, *bid*(*bob*, *lamp*))	100		
19	$\Box\neg$**Org**(mike, *viol*(*mike*, *forbidden*, *verified*(*bob*)))	154		
20	$\Box$(**Org**(bob, *viol*(*bob*, *obliged*, *verified*(*bob*))) $\rightarrow$ $\Diamond$**D**(mike, *removeAgent*(*bob*, *lamp*)))	154		

ϕ = *norm*(*sally*, *seller*, *obliged*, *verified*(*sally*), *auction_created*(*sally*, *lamp*))

become verified before bidding, Mike will eventually remove him from the auction. All of these properties require us to explore the full state space, which is possible because we only consider a small part of the scenario.

8.3.2 Scaling systems

The experiments above showed that in smaller systems it is possible to verify various properties, but as soon as the number of agents or the program size increases, the number of states greatly increases, making verification impractical or infeasible.

In this section, we investigate how much the number of agents affect the size of the state space. We do so by programming a very simple agent, which is able to enact a role and inform other agents about the enactment. We then verify a few simple properties on an increasing amount of those agents.

Each agent consists of the following Gwendolen-program, which basically serves to provide them with the capability to register. This generates the option to enact the customer role.

```
1 :name: agent
2 :Plans:
3 +!registered(Me) [achieve] : { True } <- register(address, account);
```

Furthermore, the agents use the following AORTA program to perform their organizational reasoning.

```
1 role(customer) : true => enact(customer).
2 send(true, tell, rea(Me, Role))
3   : bel(me(Me)), bel(agent(A)), ~bel(me(A)),
4     ~bel(sent(A, org(rea(Me,Role))))
5   => send(A, org(rea(Me, Role))).
```

Thus, the agents will enact the customer-role, and will inform every other agent about this enactment.

We check the following properties for an increasing number of agents to investigate how the state space increases:

$$\Diamond \bigwedge_{i \in \mathbb{A}} \mathbf{Org}(a_i, rea(a_i, customer)) \qquad \text{(Enacted)}$$

$$\Diamond \bigwedge_{\substack{i,j \in \mathbb{A} \\ i \neq j}} \mathbf{Org}(a_i, rea(a_j, customer)) \qquad \text{(Informed)}$$

We ran the verification procedure for 2 to 10 agents. As shown in Figure 8.3, the number of states increases exponentially. In the end, we did not complete the verification of the informed-property for 9 and 10 agents, because the time spend for verification with 8 agents was 17 hours, which can be partly attributed the number of agents and partly the fact that the formula becomes quite big, thus further slowing down the verification.

It should be clear from the figure that the number of states quickly reaches a level where verification is simply infeasible. Furthermore note that we are considering a quite

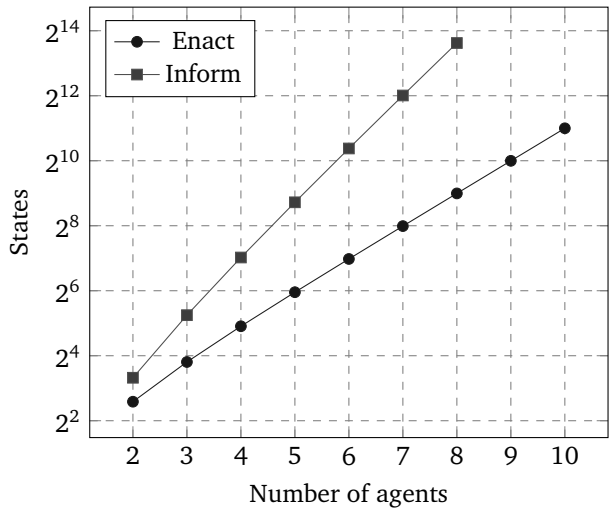


Figure 8.3: The number of states explored in two scenarios compared to the number of agents involved.

simple program with few branching points, so in more advanced programs, we would expect an even greater number of states.

8.3.3 Towards verification of "good" organizations

The LAO formalism [Dignum and Dignum, 2012], which we used in Chapter 4 to describe norms, proposes a number of key criteria that an organization should ideally meet in order to be successful. This includes e.g. the notion of a *good organization* and an *efficient organization*. The success of an organization is very closely related to the agents participating in the organization, so given an implementation of organization-aware agents in a system, it will be useful to be able to decide whether those agent are sufficient for the organization to succeed.

The following criteria are given:

Role initiative: Initiative for a role r to achieve ϕ means that an agent playing r should perform some action to achieve ϕ. This usually means either attempting to achieve ϕ itself or delegate it to someone else.

Well-defined: In a well-defined organization, each objective is related to a role that has the initiative to achieve it.

Successful: A successful organization is a well-defined organization in which some of the agents furthermore have the capability to achieve the objectives.

Good: In a good organization, role initiative for an objective means that some agent will eventually attempt to achieve the objective.

Efficient: Finally, in an efficient organization, each agent knows who to delegate its tasks to, if it is incapable of achieving them itself.

We can see from the description above that agent capability is an essential part of proving these organizational properties. We thus further extend the PSL syntax to incorporate agent capability[5]:

$$\psi' ::= \psi \mid \mathbf{C}(ag, f)$$

The interpretation of $\mathbf{C}(ag, f)$ is given as:

$$MAS \models \mathbf{C}(ag, f) \quad \text{iff} \quad f \in \mathsf{cap}(ag),$$

where $\mathsf{cap}(ag)$ is the agent's capabilities as defined in Chapter 5.

We can then specify the properties in PSL-like syntax as follows. First, we define role initiative:

$$I_r\phi \equiv \exists a \in \mathbb{A} : (\mathbf{Org}(a, \mathsf{rea}(a, r)) \wedge \Diamond(\mathbf{I}(a, \phi) \vee \exists b \in \mathbb{A} : \mathbf{D}(a, \mathsf{send}(b, \mathbf{opt}(\phi))))).$$

That is, a role r has the initiative to complete ϕ iff there is an agent playing the role, which eventually either intends to complete the objective[6] *or* delegates the objective to another agent.

We can then define specify the properties as follows (where ψ denotes an action that results in ϕ):

$$\forall a \in \mathbb{A} : \Box(\mathbf{Org}(a, \mathsf{obj}(\phi, O)) \rightarrow \exists r : (\mathbf{Org}(a, \mathsf{role}(r, O)) \wedge I_r\phi)) \qquad \text{(Well-defined)}$$

$$\begin{aligned} &\forall a \in \mathbb{A} : \Box(\mathbf{Org}(a, \mathsf{obj}(\phi, O)) \\ &\qquad \rightarrow \exists r : (\mathbf{Org}(a, \mathsf{role}(r, O)) \wedge I_r\phi \\ &\qquad\quad \wedge \exists a' \in \mathbb{A} : (\mathbf{Org}(a', \mathsf{rea}(a', r)) \wedge \mathbf{C}(a', \phi)))) \end{aligned} \qquad \text{(Successful)}$$

$$\Box(I_r\phi \rightarrow \exists a \in \mathbb{A} : (\mathbf{Org}(a, \mathsf{rea}(a, r)) \wedge \Diamond\mathbf{D}(a, \psi))) \qquad \text{(Good)}$$

$$\begin{aligned} &\Box((I_r\phi \wedge (\forall a \in \mathbb{A} : \mathbf{Org}(a, \mathsf{rea}(a, r)) \rightarrow \neg\mathbf{C}(a, \phi)) \\ &\qquad \wedge \exists q : (\mathbf{Org}(a, \mathsf{role}(q, O)) \wedge \mathbf{Org}(a, \mathsf{dep}(r, q, \phi)) \\ &\qquad\quad \wedge (\exists b \in \mathbb{A} : \mathbf{Org}(a, \mathsf{rea}(b, q))))) \\ &\qquad\quad \rightarrow \Diamond\exists b : (\mathbf{Org}(a, \mathsf{rea}(b, q)) \wedge \mathbf{G}(b, \phi))) \end{aligned} \qquad \text{(Efficient)}$$

A *well-defined* organization is thus one where for each objective, there exists a role which has the initiative to achieve it. Note that we universally quantify the agent to verify that all the agents believe that the objective can be achieved (and is thus well-defined). *Success* is similar, but further requires that there exists an agent enacting the role with the capability

[5] Even though capabilities are static in Gwendolen, this need not be the case for other APLs (consider, e.g., *Jason*), so reasoning about capabilities in a temporal setting can be quite useful – even outside the organizational setting.

[6] We thus assume that by intending to complete the objective, the agent will eventually *attempt* to complete it.

to achieve the objective. In a *good* organization, at least one agent with the initiative to complete a role objective will eventually attempt to achieve the objective (by executing an action which is known to achieve the objective). Finally, we use the dependency relation to define an *efficient* organization to verify that the agents can use the organizational structure to delegate their objectives.

While the properties capture our understanding of each notion, we cannot directly specify general properties like this in PSL since only ground formulas can be used. Instead, they must be specified using specific formulas relevant to the given system. This leads to rather large formulas and since the time complexity of LTL model checking is exponential in the length of the formula [Baier and Katoen, 2008], it is generally not feasible to verify these kinds of formulas in AJPF.

We can, however, decide which parts of the formulas are relevant and handpick those for verification. While not decisively proving the organization's success, it enables us to verify that (at least) parts of the organization is, e.g., well-defined or efficient.

Consider, e.g., the objective to verify customers. We can verify that (agents enacting) the seller-role has the initiative to verify agents by using the following formula:

$$\begin{aligned}&\mathbf{Org}(\texttt{sally}, \mathsf{rea}(\mathit{sally}, \mathit{seller}))\\ &\quad \wedge\ \Diamond(\mathbf{I}(\texttt{sally}, \mathit{verified}(\mathit{sally})) \vee \mathbf{D}(\texttt{sally}, \mathsf{send}(\mathit{mike}, \mathbf{opt}(\mathsf{obj}(\mathit{verified}(\mathit{sally}))))))\end{aligned}$$

That is, Sally has the initiative to verify herself if she either intends to verify herself, or she delegates it to Mike. We can furthermore verify that the organization is good (with regards to verification of customers):

$$\begin{aligned}\Box((\mathbf{Org}(\texttt{mike}, \mathit{rea}(\mathit{mike}, \mathit{manager})) \wedge \mathbf{I}(\texttt{mike}, \mathit{verified}(\mathit{sally})))\\ \rightarrow \Diamond\mathbf{D}(\texttt{mike}, \mathit{verify}(\mathit{sally})))\end{aligned}$$

Here, we have omitted a few irrelevant parts of the formula (i.e., the fact that Mike can delegate the task to someone else, and the role enactment proposition in the consequent of the implication). The property states that if Mike enacts the manager-role and has the intention to verify Sally, he will eventually perform the action that verifies her.

We verified the properties in the simple scenario. The first property required us to search 294 states, and in the second, which searches the entire state spaces, we searched 1306 states.

It is thus possible to perform this kind of verification in a small setting, but then it does not really verify that the organization in its entirety is successful, good or efficient. Furthermore, each formula must be translated into a domain-specific version with ground formulas, which, if done manually, is a tedious process, and otherwise will often generate quite large formulas, since all agents, roles and objectives must be considered.

8.4 Concluding Remarks

In this chapter, we evaluated the AORTA system by means of *model checking*. That is, by using a technique that exhaustively searches through the system, we were able to verify certain properties of systems using AORTA.

We used the auction house scenario to show that properties about organizational structure, organizational options and norms can be verified. Among other things, we showed that we can verify that when a norm is violated, proper measures will be taken by the relevant agents. The greatest drawback of the approach is that it is slow: in order to verify rather simple properties of the system, we had to reduce the number of agents and remove parts of the scenario.

We furthermore investigated how the number of agents affect the number of states in the system, and we showed that more general properties about an organization can be specified, though it is a somewhat tedious process. To make this practically useful, more work is required to reduce the state space (perhaps using state matching), to allow verification of larger systems.

PART FOUR

DECIDING BETWEEN INFLUENCES

Having proposed a framework for organizational reasoning that allow agents to commit to and complete organizational objectives, a natural follow-up question is that of decision-making: How does the agent decide whether to follow the expectations of the organization or its own desires?

We propose a consequence-based approach to decision-making that allows agents to consider the expected consequences of their actions (for example based on the expectations from the organization) before deciding what to do. We present our formal model, which is based on qualitative decision theory (Chapter 9), and we show a proof-of-concept implementation of the model, integrated with AORTA and the *Jason* agent programming language (Chapter 10).

CHAPTER 9

Consequence-Based Decision-Making

In the previous chapters, we have built a framework that allows agents to reason about organizations. Since BDI agents are characterized by their mental attitudes, we know that their decisions are based on what they believe, desire and intend to do. Agents are therefore not a priori well suited for basing their decisions on external influences, unless those influences coincide with their desires. For example, why should the agent choose to become verified when entering an auction? Why should the agent pay for items that it has won? Evidently, the reason for following such external influences is that otherwise, the agent might not be able to complete its own objectives. However, how can it reason that this is the case? We argue that the underlying reason is that the agent cannot *tolerate* choosing not to commit to these external influences: the consequences of doing so are undesirable.

This part distinguishes itself from the rest of the thesis in that the focus is not on organizational reasoning as such. Instead, we look at the agent as an intelligent entity that is influenced by different sources (e.g., the AORTA-component), and we discuss how it can decide which of the influences to commit to, especially when the influences are conflicting. The work is based on [Jensen, 2013a], but has been extended in a number of ways:

- The description of the decision procedure and the algorithms for model generation has been extended and optimized (this chapter).
- We show a clear relation between expectations in this model and norms in the AORTA metamodel (Chapter 10).
- We describe a proof of concept implementation of the system in Java, that can be integrated in APLs to perform decision-making based on the notion of influences (Chapter 10).

When agents are constrained by an organization, their own goals may conflict with those of the organization and they need in such cases to be able to decide which of the conflicting goals to pursue. As mentioned in Chapter 3, in some of the previous work toward resolving such conflicts, desires and obligations are ordered a priori, so that an agent either prefers desires over obligations or obligations over desires. This results in agents that are always selfish (considering own goals more important than organizational goals) or always social (vice versa). We argue that such distinction can be too hard; even a selfish agent could in some cases benefit from preferring certain obligations to its desires. We propose a model for resolving such (and other) conflicts, based on work in the area of qualitative decision theory by Boutilier [1994], in which we consider the expected consequences of bringing about a state. We show that this result in agents that are not

always either social or selfish, but instead are able to decide based on the context and the consequences of bringing about a state.

Our focus is on a general approach toward deciding between different kinds of influences, with the aim to show that although agents are subject to influences from different entities, they are able to make decisions based on the current situation, their preferences and the expected consequence of bringing about a state. We do not focus explicitly on the choice between an agent's desires and the obligations from an organization, but emphasize that the approach is useful in this situation and other situations as well.

The chapter is organized as follows. In Section 9.1, we discuss the issues that arise when an agent has to make a decision between conflicting influences. In Section 9.2, we present a new approach on how to solve such conflicts without having to put the agents into the categories "selfish" or "social". We present a method for generating models that conform to the agent's preferences in Section 9.3. Finally, in Section 9.4, we discuss a case in which agents have conflicting influences and show that our method enables them to make a decision using their own preference and the expected consequence of bringing about each state.

9.1 Conflicting Influences

Agents entering an environment will be subject to influences from multiple sources: e.g., their own desires, other agents and organizations. Since desires become intentions once committed to, one could argue that by committing to influences from other sources, these influences are merely desires as well, i.e., the agent simply desires to do so. The incentives for committing to external influences are however not clear, since there should be different reasons for committing to internal influences (desires) and to external influences (e.g., obligations). For example, if an agent has a desire to buy something from an auction, it typically *wants* to do so. However if the agent wants to become verified before bidding, this "desire" is most likely caused by the fact that the agent does not want to be sanctioned, rather than being an actual desire to become verified. In such a situation, the agent should reason differently, since the (implicit) desire is actually to avoid being sanctioned.

Furthermore, consider an agent that receives an undesirable request from a friendly agent. It may choose to commit to the task even though the task itself is not desirable, because the desire to help the other agent is stronger than the desire not to perform the task (the consequence of *not* helping the other agent might be a bad reputation). Similarly, if an agent is obligated to perform certain tasks for an organization, it should be able to consider not only whether the tasks are desirable, but also weigh this against the penalty for violating the obligation.

In what follows, we call something that the agent might choose to intend to do a "decision influence" rather than a desire, since it, as argued above, may stem from many different sources rather than being merely desires. The agent naturally has to consider its desires since it would be irrational to ignore them, but since ignoring an obligation

might lead to an intolerable state (i.e., a sanction), the agent has to consider obligations as influences as well. This means two things:

1. The agent should not solely consider whether it wants to bring about some state (i.e., whether that state, compared to other states, is most preferred), but should also consider whether the expected consequences of bringing about the state are tolerable.
2. The agent should not be concerned with whether each influence is a desire, an obligation or something else; they are considered as equal influences, and only their preference and consequences should be taken into account – not their type.

We emphasize that when we talk about goals and desires, we refer to the notion of achievement-goals, i.e., a goal (or desire) to achieve a state where ϕ is true. Thus, when we talk about the preference of having a goal ϕ, we mean the preference of being in a state where ϕ is true.

Several approaches are proposed on how to let agents choose between specific types of influences (typically obligations and desires) [Broersen et al., 2001; Dignum et al., 2000; Dignum et al., 2002]. We have already discussed this work in Chapter 3 in relation to organizational reasoning, so we focus on how the approaches handle various types of influences.

The BOID architecture [Broersen et al., 2001], which focus on conflicts between the mental attitudes beliefs, obligations, intentions and desires, proposes a strict ordering between these attitudes, such that the order of derivation determines the agent's attitude. In this way, different agent types emerge; an agent deriving desires before beliefs is a wishful-thinking agent, while an agent deriving obligations before desires is social.

However, we believe that such ordering is too strong: if an agent is strictly social, it will always choose obligations over desires, and vice versa for selfish agents. This might not always be appropriate. For instance, a selfish agent might desire not to go to work, but if the consequence of not fulfilling the obligation of going to work is severe (i.e., being fired), even a selfish agent should consider this consequence before deciding not to go to work. In other words, a selfish agent should be able to behave as a social agent *if this benefits the agent*.

Dignum et al. [2000] suggest that *"both norms and obligations should be explicitly used as influences on an agent's behavior"*. By modifying the BDI-interpreter, it becomes possible to consider deontic events, such that the agents can deliberate over, e.g., obligations. The modified interpreter generates a number of options depending on these events and chooses a relevant plan based on the agent's attitude. In this case, they consider the influences by their type and handle them as such. They do argue, however, in [Dignum et al., 2002] that the preference orderings induced by desires, obligations and norms should be combined into a single ordering. It is noted that a common way to do so is to allow that a single preference ordering determine the aggregate ordering, such that the agent might always put obligations over norms and norms over desires, similarly to the BOID architecture.

They further discuss another approach, which perhaps more closely resembles ours, in that they map the orderings into a common scale, such that very desirable situations could outweigh the cost of violating certain obligations. In that way, a selfish agent might choose to go to work, because the cost of being fired is too high, compared to the utility gained from staying at home.

Of course, the notion of social and selfish agents is just one classification of agents in societies. Conte and Sichman [1995] classify social agents in heterogeneity and complementarity dimensions, where heterogeneous agents have different goals and abilities, while two or more agents can have complementary actions, meaning that they can be used to jointly achieve some goal. These notions classify agents in a different way with a focus on ability, and could complement the notions of being social and selfish. For example, a social agent with complementary abilities might be more inclined to cooperate compared to a social agent without these abilities. This is however not captured by our model and is out of scope for this work, but it is definitely worth noting as an alternative approach to decision-making.

9.1.1 Consequence-based decisions

Performing an action will in many cases result in one or more side effects that may or may not be desirable for the agent performing the action. These side effects are part of the consequences of performing the action, so if the agent can use this information by considering the expected consequences, it will be able to make better decisions. In other words, to reason about bringing about a certain state, the agent should consider what consequences are expected when bringing about that state.

We therefore suggest that the agent should reason about the expected *consequences* of choosing to commit to a decision influence and furthermore that this reasoning should be based on the notions of *preference* and *tolerance*:

- From the set of influences, the agent can choose those that are most **preferred**. This could be some of its own desires, but could also be obligations that for some reason are preferable (e.g., if they coincide with the agent's overall objectives).
- Bringing about a preferred state may however lead to unintended (and undesirable) consequences. For example, if I choose to stay at home, I risk being fired. We thus propose to consider whether the consequences of bringing about a state are **tolerated**.

The reason for using tolerance instead of preference in the case of consequences is that the agent should not need to desire the consequences of bringing about a state. Since the consequences are merely side effects, they need not be desired in the same way as the influences are. If a consequence is preferable, then clearly it is also tolerable but the opposite need not be the case (the agent might tolerate going to work even though it *prefers* to stay at home). We define a situation as being tolerated when the opposite is not preferred (e.g., working is tolerated if *not working* is not preferred over working).

Using the influences, we can build two sets to base a decision on: the set of most preferred influences, *Pref*, and the set of influences with the most tolerable expected consequences, *Tol*. From this, we can identify different strategies for how to make a decision based on these sets:

$$Pref > Tol \tag{9.1}$$
$$Pref < Tol \tag{9.2}$$
$$Pref \cup Tol \tag{9.3}$$
$$Pref \cap Tol \tag{9.4}$$

We can let preferences take precedence (9.1) such that if a single influence is most preferred (in *Pref*) it is chosen, and only in case of multiple most preferred influences will tolerance be taken into account, or we can let tolerance take precedence (9.2), which gives the opposite situation. However, this means that only in some cases are both preference and tolerance taken into account, so an agent could choose, e.g., to commit to something it prefers which leads to an intolerable situations that could have been avoided if both sets were taken into account. We could take the collective influences (9.3), but then the agent would have to choose between things it prefer and things it tolerate even though the former may be intolerable and the latter could be unwanted. Finally, we could let the decision be the influences that are *both* preferred and tolerated (9.4), thus ensuring that the decision is preferred by the agent and that the consequences can be tolerated. In certain situations, these sets may not coincide, and we argue then that the safest decision is to choose something tolerated, since even though the influence might not be *most* preferred, at least it will not lead to an intolerable state. Our approach makes use of the last strategy, i.e., taking the intersection of the sets, since it incorporates both measures in all situations, while not resulting in intolerable states.

Note that our approach does not incorporate an explicit notion of organizations; the focus is on many different kinds of influences including the obligations toward an organization. As a result, we consider consequence as expectations from the environment, and model it by an ordering of possible states. We can then reason about which states are most expected given that something has happened. This means that if the consequences of the violation of an obligation (i.e., sanctions) are specified in an organizational model, these consequences are in our approach modeled such that worlds in which the violation has occurred *and* a sanction has been imposed are more expected than the worlds where the violation has occurred without the agent being sanctioned. We use this in Chapter 10 to show that the conditional norms of the AORTA metamodel can be used to specify the expectations.

9.2 Modeling Influence and Consequence

We base our work on the Logic for Qualitative Decision Theory (QDT) by Boutilier [1994]. We briefly describe the semantics of QDT and define a few new abbreviations to be used in the decision-making.

The basic idea behind the QDT model is as follows. An agent has the ultimate desire of achieving the goals to which it is committed. This can be modeled by a possible worlds-model in which the agent has achieved its goal when it is in a world where those goals hold. The most preferred world in an ideal setting is the world in which the agent has achieved all of its goals. However, such world is often unreachable since the agent could have contradicting goals, other agents could prevent the agent from achieving all of its goals, an organization could impose obligations, which contradict the agent's goals, etc. By ordering the worlds in a preference relation, it is possible to choose the most preferred world(s) in a sub-ideal situation.

In order to decide between influences, the consequence of bringing about a state should be taken into account. If the consequence of pursuing a personal desire is to be fired from your workplace, it might not be reasonable to do so even though the desire was more preferred than the obligations from work.

A QDT model is of the form:

$$M = \langle W, \leq_P, \leq_N, \pi \rangle,$$

where W is the non-empty set of worlds, $\leq_P$ is the transitive, connected preference ordering[1], $\leq_N$ is the transitive, connected normality ordering, and π is the valuation function. The normality ordering is used to model how likely each world is (e.g., agents are normally verified when bidding on an item), and the preference ordering is used to model an agent's preferences.

The semantics are as follows:

$$\begin{aligned}
M, w \models p &\iff p \in \pi(w) \\
M, w \models \neg\varphi &\iff M, w \not\models \varphi \\
M, w \models \varphi \wedge \psi &\iff M, w \models \varphi \wedge M, w \models \psi \\
M, w \models \Box_P \varphi &\iff \forall v \in W, v \leq_P w, M, v \models \varphi \\
M, w \models \overleftarrow{\Box}_P \varphi &\iff \forall v \in W, w <_P v, M, v \models \varphi \\
M, w \models \Box_N \varphi &\iff \forall v \in W, v \leq_N w, M, v \models \varphi \\
M, w \models \overleftarrow{\Box}_N \varphi &\iff \forall v \in W, w <_N v, M, v \models \varphi
\end{aligned}$$

where $\Box_P$ and $\overleftarrow{\Box}_P$ are modal operators for preference, and $\Box_N$ and $\overleftarrow{\Box}_N$ are modal operators for normality. $\Box_P \phi$ thus holds in some world w if ϕ holds in all worlds that are at least preferred as w, whereas $\overleftarrow{\Box}_P \phi$ holds if ϕ holds in all worlds less preferred than w (similarly for normality).

We can define the other operators ($\vee, \rightarrow, \Diamond, \overleftarrow{\Diamond}$) as usual. Finally, we can talk about a formula being true in all worlds or some worlds: $\overleftrightarrow{\Box}_P \varphi \equiv \Box_P \varphi \wedge \overleftarrow{\Box}_P \varphi$ and $\overleftrightarrow{\Diamond}_P \varphi \equiv$

[1] We adopt the notion by Boutilier and others that we prefer minimal models, so $v \leq_P w$ denotes that v is at least as preferred as w.

$\Diamond_P \varphi \vee \overleftarrow{\Diamond}_P \varphi$, respectively (similarly for normality). The following abbreviations are defined [Boutilier, 1994]:

$$
\begin{array}{lrcll}
(1) & I(\psi \mid \varphi) & \equiv & \overleftrightarrow{\Box}_P \neg\varphi \vee \overleftrightarrow{\Diamond}_P(\varphi \wedge \Box_P(\varphi \rightarrow \psi)) & \text{(Conditional preference)} \\
(2) & \varphi \leq_P \psi & \equiv & \overleftrightarrow{\Box}_P(\psi \rightarrow \Diamond_P \varphi) & \text{(Relative preference)} \\
(3) & T(\psi \mid \varphi) & \equiv & \neg I(\neg\psi \mid \varphi) & \text{(Conditional tolerance)} \\
(4) & \varphi \Rightarrow \psi & \equiv & \overleftrightarrow{\Box}_N \neg\varphi \vee \overleftrightarrow{\Diamond}_N(\varphi \wedge \Box_N(\varphi \rightarrow \psi)) & \text{(Normality conditional)}
\end{array}
$$

The abbreviations state that (1) ψ is ideally true if φ is true, (2) φ is at least as preferred as ψ, (3) ψ is tolerable given φ and (4) that ψ normally is the case when φ is.

In order to make decisions as motivated above, we define the following abbreviations, which allow us to specify different kinds of relative preference, and relative tolerance.

$$
\begin{array}{rcll}
\varphi \not\leq_P \psi & \equiv & \neg(\varphi \leq_P \psi) & \text{(Not as preferred)} \\
\varphi \approx_P \psi & \equiv & (\varphi \leq_P \psi \wedge \psi \leq_P \varphi) & \\
 & & \quad \vee (\varphi \not\leq_P \psi \wedge \psi \not\leq_P \varphi) & \text{(Equally preferred)} \\
\varphi \leq_{T(\gamma)} \psi & \equiv & (T(\varphi \mid \gamma) \wedge \neg T(\psi \mid \gamma)) \vee & \\
 & & \quad ((T(\varphi \mid \gamma) \leftrightarrow T(\psi \mid \gamma)) \wedge & \\
 & & \quad (\varphi \leq_P \psi \vee \varphi \approx_P \psi)) & \text{(Relative tolerance)}
\end{array}
$$

Relative tolerance is defined as φ being at least as tolerable as ψ with regards to γ when either φ is tolerable given γ and ψ is not, or both φ and ψ are tolerable given γ (or both are not), and φ is at least as preferred as ψ, or they are equally preferable. This means that even if neither is tolerable, they are still comparable.

9.2.1 Making a decision

We now proceed to show how QDT can be used to decide between conflicting influences. We define a model for an agent's decision-making as follows:

$$\mathscr{M} = \langle M, F, C, \Sigma \rangle,$$

where M is a QDT-model as defined above, F is the set of influences, C is the set of controllable atoms, and Σ is the agent's belief base.

A controllable atom is, roughly, an atom that the agent is able to influence, directly or indirectly, by an action. For example, Bob cannot control whether Sally registers, so *register*(*sally*) is not controllable by Bob (though it is by Sally) and cannot be a consequence of one of his actions. On the other hand, *registered*(*bob*) is, and the potential consequences of this should thus be considered. Furthermore note that atoms such as *verified*(*bob*) are *indirectly* controllable, because even though Bob cannot perform the verification process himself, he can perform actions that lead to Mike doing it (i.e., by sending a message that delegates the task).

Definition 9.1 (Potential consequence)**.** The set of potential consequences *Pot* is defined such that if $\varphi \in C$ then $\varphi, \neg\varphi \in Pot$. That is, if φ is controllable, then one of $\varphi, \neg\varphi$ may be a consequence of bringing about some state.

In order for a potential consequence to be an actual (expected) consequence of φ, it has to follow from the most normal worlds where φ holds. That is, we add φ to the belief base Σ, and the potential consequences that follow from the expanded belief base are considered the expected consequences.

When doing so, we have to take a number of things into account:

- Is the negation of the potential consequence, C_φ, in Σ? If so, it must be removed so we can check whether C_φ normally follows from Σ. Naturally, this cannot be the case if $\neg C_\varphi \in \Sigma$, but since C_φ is a controllable atom, we have to consider whether achieving φ normally leads to C_φ.
- Are φ and Σ consistent? If not, we have to take care when adding φ to ensure that Σ is consistent afterwards.

The first point can be accommodated by letting $\Sigma' = \Sigma \setminus \{\neg C_\varphi\}$. We are then able to consider whether φ leads to C_φ.

For the second point, let us assume that φ and Σ are consistent. We can then add φ to Σ using the *expansion* operator, $+$, of the AGM theory [Alchourrón et al., 1985], where $\Sigma + \varphi$ means adding φ to a copy of Σ and closing the resulting set under logical consequence. We work with a copy of the belief base since our reasoning concerns what happens *if* the literal is added.

If, however, φ and Σ are not consistent, we can use the AGM *revision* operator, $\dot{+}$, which behaves like $+$, but minimally modifies Σ to make it consistent with φ, before adding φ. For example, if $\Sigma = \{a, b\}$, and $\phi = \neg a$, the belief base must be revised to add ϕ. We require this, since otherwise, we cannot reason about what happens *if* ϕ is the case.

Definition 9.2 (Expected consequences)**.** Given an agent's (revised) belief base Σ', the set of potential consequences *Pot*, a literal φ and a QDT-model M, the expected consequences of bringing about φ, denoted $EC_M(\varphi)$, are given by:

$$EC_M(\varphi) = \bigwedge \{C_\varphi \mid (\Sigma' \dot{+} \varphi \Rightarrow C_\varphi) \text{ where } C_\varphi \in Pot\}$$

That is, the expected consequences are defined as the conjunction of all literals C_φ that are normally consequences of the current belief base Σ' expanded with φ, such that Σ' remains consistent. If there are no expected consequences, then $EC_M(\varphi) = \top$. In what follows, we omit the model, i.e., write $EC(\varphi)$ when the intended meaning is clear.

Consider a normality ordering in which we have that

$$a \wedge x \Rightarrow b, \qquad a \wedge \neg x \Rightarrow c, \quad d \wedge \neg x \Rightarrow e,$$

and belief base $\Sigma = \{x\}$. We then have that $EC(a) = b$ and $EC(d) = \top$. If $\Sigma = \{\neg x\}$, then $EC(a) = c$ and $EC(d) = e$.

We can now formally define the sets of most preferred influences and most tolerable consequences.

Definition 9.3 (Most preferred influences)**.** Given an agent's set of influences F, the most preferred influences are defined as the set *Pref*:

$$Pref = \{\varphi \mid \varphi \in F \wedge \forall \psi \in F \ (\psi \neq \varphi \rightarrow \varphi \leq_P \psi)\}$$

Definition 9.4 (Most tolerable consequences)**.** Given an agent's set of influences F, the influences with the most tolerable consequences are defined as the set *Tol*:

$$Tol = \{\varphi \mid \varphi \in F \wedge \forall \psi \in F \ (\psi \neq \varphi \rightarrow EC(\varphi) \leq_{T(\varphi \vee \psi)} EC(\psi))\}$$

An agent can use these sets to decide its course of action by selecting the most preferred influences having the most tolerable consequences from the set of potentially conflicting influences.

Definition 9.5 (Decision)**.** Given a the set of influences F and the expected consequences $EC(\varphi)$ for all $\varphi \in F$, the set of best influences (the decision) the agent should choose from, *Dec*, is defined as follows:

$$Dec = \begin{cases} Tol & \text{if } Tol \cap Pref = \emptyset \\ Tol \cap Pref & \text{otherwise} \end{cases}$$

Given a model, $\mathscr{M}$, an agent can then choose an arbitrary literal from *Dec*, since all of these will be preferred and have tolerable consequences (or at least have tolerable consequences).

If there are no expected consequences of bringing about a certain state, i.e., if $EC(\varphi) = \top$, then φ is considered tolerable since we do not expect any consequences. Therefore comparing the relative tolerance for all other consequences, ψ, is reduced to comparing $\top \leq_{T(C)} \psi$ and $\psi \leq_{T(C)} \top$. Note that $T(\top \mid \psi)$ is true iff ψ is true in any world[2]. Furthermore, $\top \leq_P \psi$ is always true, and $\psi \leq_P \top$ is true iff ψ is true in all worlds. Thus, it is possible to make a decision even if some influences have no known consequences.

[2]Since $T(\top \mid \psi) \equiv \overleftrightarrow{\Diamond}_P \psi \wedge \overleftrightarrow{\Box}_P (\neg\psi \vee \Diamond_P (\psi \wedge \top)$.

As we will show below, the procedure is designed such that given a non-empty set of influences, the set of best influences is non-empty. To do this, we first prove the following lemma concerning relative tolerance.

Lemma 9.6. *Given expressions φ, ψ, and γ, the following relation holds for relative tolerance:*

$$\neg(\varphi \leq_{T(\gamma)} \psi) \rightarrow (\psi \leq_{T(\gamma)} \varphi).$$

That is, given two states, one of them will be at least as tolerable as the other.

Proof. We assume $\neg(\varphi \leq_{T(\gamma)} \psi)$ and prove that $(\psi \leq_{T(\gamma)} \varphi)$. Based on the assumption and the definition of relative tolerance, the following formulas hold:

$$\neg(T(\varphi \mid \gamma) \wedge \neg T(\psi \mid \gamma)) \tag{9.5}$$

$$\neg((T(\varphi \mid \gamma) \leftrightarrow T(\psi \mid \gamma)) \wedge (\varphi \leq_P \psi \vee \varphi \approx_P \psi)) \tag{9.6}$$

1. Given (9.5), we have that either $T(\varphi \mid \gamma) \leftrightarrow T(\psi \mid \gamma)$ or $\neg T(\varphi \mid \gamma) \wedge T(\psi \mid \gamma)$ holds. In the latter case we have that $\psi \leq_{T(\gamma)} \varphi$ by the definition of relative tolerance. Otherwise, they are equally tolerable and we have to consider the second case.
2. Given (9.6), either $\neg(T(\varphi \mid \gamma) \leftrightarrow T(\psi \mid \gamma))$ or $\neg(\varphi \leq_P \psi \vee \varphi \approx_P \psi)$. If the former is the case, then one is tolerated and the other is not. Because of (9.5), we have that $\neg T(\varphi \mid \gamma) \wedge T(\psi \mid \gamma)$ and therefore $\psi \leq_{T(\gamma)} \varphi$. If the latter is the case then we have that $\neg(\varphi \leq_P \psi) \wedge \neg(\varphi \approx_P \psi)$. In that case we have that $\psi <_P \varphi$ and therefore $\psi \leq_{T(\gamma)} \varphi$.

□

Proposition 9.7. *Given a non-empty set of influences F and the expected consequence $EC(\varphi)$ for each $\varphi \in F$, the set of best influences, Dec, is always non-empty.*

Proof. If $|F| = 1$ then $Dec = F = Tol = Pref$, since there are no $\psi \neq \varphi$ in F. If $|F| > 1$ then we consider each case.

- If $Tol \cap Pref = \emptyset$ then $Dec = Tol$ and we have to show that $Tol \neq \emptyset$. If $Tol = \emptyset$ then there is no ψ such that $EC(\varphi)$ is relatively more tolerated than $EC(\psi)$. Since $|F| > 1$ there is at least one $\psi \neq \varphi$, and by Lemma 9.6 we then have that $EC(\psi)$ is relatively more tolerated than $EC(\varphi)$. Thus, $\psi \in Tol$ and $Dec \neq \emptyset$.
- If $Tol \cap Pref \neq \emptyset$ then, since $Dec = Tol \cap Pref$, Dec cannot be empty.

□

Proposition 9.7 shows that the decision procedure will always produce a non-empty result. This means that agents can use the procedure even in situations where there is no conflict between influences. For example, the fact that the organization expects the agent to verify before bidding does not conflict with having a desire to bid on an item, but as shown above the procedure can still be used to decide, e.g., that verification should be done before bidding (to avoid sanctions).

9.3 Generating Models

The preferences of an agent are usually not described as a model as shown above, but will rather be expressions such as "I prefer to bid only on items I want" or "when I participate in an auction I do not want to be sanctioned". In order to utilize such preferences in the decision procedure above, a transformation is required. In the following, we present a method, which will generate a QDT-model that respects non-contradictory rules specified by the agent.

Each agent specifies a set of rules of the form $\varphi \rightsquigarrow \psi$, where φ and ψ are standard first-order ground formulas. A rule, $\varphi \rightsquigarrow \psi$, should be read as "if φ then normally/preferably ψ". Using the notion of possible worlds, we understand a rule as follows. Worlds w, in which $w \models \varphi \wedge \psi$, are favored over worlds w', where $w' \models \varphi \wedge \neg\psi$. Thus, a rule is roughly interpreted as the conditionals for preference and normality. We propose a method for generating preference and normality orderings that respect such rules by utilizing this interpretation. The generic definition of the conditional operators from the previous section is:

$$\varphi \rightsquigarrow \psi \equiv \overleftrightarrow{\Box}\neg\varphi \vee \overleftrightarrow{\Diamond}(\varphi \wedge \Box(\varphi \rightarrow \psi)).$$

From this definition, it is clear that there are two ways to ensure that a rule $\varphi \rightsquigarrow \psi$ is respected. Either (a) φ is never true or (b) in the most favored world(s) where φ is true, ψ is also true. Option (a) is achieved easily; we simply remove all worlds where φ is true. However, the agent does probably not intend this, since the rules are most likely specified such that favored situations are also possible situations. We therefore require that the method does not remove any worlds from W. The method should furthermore ensure that after applying a rule, the rule holds (i.e., $M \models \varphi \rightsquigarrow \psi$). Another natural requirement is that previously applied rules should still hold after application of a new rule. If this is not possible, we say that the new rule contradicts previously applied rule, and therefore discard the new rule.

The aim is to generate a model respecting the rules, such that the agent can make a decision based on the model. This is done by performing a number of steps:

1. Generate the possible worlds based on the rules, influences and controllable atoms.
2. Create an initial ordering of worlds.
3. Apply each rule to the ordering.

Rather than generating a model in which all rules are applicable, we create a submodel for the relevant part of the world given the current influences F, rules R and controllable atoms C. For example, an agent's preference concerning work might not be relevant for decisions in a different context. Furthermore, certain situations are not deemed possible, such as leaving work early and not going to work at all.

Algorithm 9.1 retrieves the relevant atoms from F and the set of rules. $\textsc{positive}(S)$ is the set of all atoms in S with all negative literals made positive, such that if $\neg\varphi \in S$ then $\varphi \in \textsc{positive}(S)$. $\textsc{atoms}(\varphi, R)$ returns a set of all atoms that appear in r where φ also appears (e.g., if $r = (\varphi \wedge \psi_1) \rightsquigarrow \psi_2$ then $\textsc{atoms}(\varphi, r) = \{\psi_1, \psi_2\}$. Notice that only rules

Algorithm 9.1 Atom retrieval

```
function RETRIEVE_ATOMS(F, C, R)
    At ← POSITIVE(F)
    checked ← ∅
    for all φ ∈ At \ checked do
        for all (ψ_a ⇝ ψ_b) ∈ R do
            if ψ_b ∈ C then
                At ← At ∪ ATOMS(φ, (ψ_a ⇝ ψ_b))
        checked ← checked ∪ {φ}
    return At
end function
```

with a *controllable* consequent are applicable. This follows from the fact that if an atom is not controllable, it cannot be a consequence of the agent's actions. Rules that lead to uncontrollable atoms will not influence the agent's decisions and are thus irrelevant.

Given the set of relevant atoms, At, the set of possible worlds, W, contains a world for each set in 2^{At}. For instance, given $At = \{a, b\}$, the initial model will be $2^{At} = \{\{a, b\}, \{\neg a, b\}, \{a, \neg b\}, \{\neg a, \neg b\}\}$. We can specify a number of impossible worlds using a set of formulas, I, e.g., $I = \{\neg a \wedge b\}$, stating that worlds where a is false and b is true are impossible. A world that entails such an expression is removed from W, and the result is then the set of possible worlds given F.

The relation between possible worlds is made by an *ordering*, $\leq$, which is the result of a mapping from a world to a natural number called the o-value, denoted $o : W \to \mathcal{N}$, such that worlds with higher numbers are more favored. Worlds have the same o-value if they are equally favored. The maximum o-value of an ordering $\leq$ is denoted $max(\leq)$. Thus, if $o(w_1) > o(w_2)$, then w_1 is more favored than w_2 and they are ordered as such in $\leq$. Initially, all worlds are equally favored, thus the initial ordering of the worlds is flat.

To guarantee that previously applied rules still hold, we propose using a *locking* mechanism in which the ordering between two worlds can be locked, such that if $lock(w_1, w_2)$ then it must always be the case that $o(w_1) > o(w_2)$. We use this to lock the ordering between worlds $w_1 = \{\varphi, \psi\}$ and $w_2 = \{\varphi, \neg\psi\}$ if a rule $\varphi \rightsquigarrow \psi$ is applied by creating a lock, $lock(w_1, w_2)$, such that w_1 is always favored over w_2. Then if a rule $\varphi \rightsquigarrow \neg\psi$ is applied, the ordering cannot be changed so that w_2 is favored over w_1 because it would result in the previously applied rule no longer being respected (since ψ would not be entailed by the most favored world where φ holds).

Rules are applied using the function APPLY : $(R, W, \leq) \to \{\top, \bot\}$ (Algorithm 9.2). Applying a rule $\varphi \rightsquigarrow \psi$ is done by finding all worlds in which both φ and ψ holds (the sought worlds) and all worlds in which φ and $\neg\psi$ holds (the contradictory worlds). The sought worlds are given an o-value of $max(\leq)+1$ and all contradictory worlds are locked in relative position to the sought worlds.

A rule $\varphi \rightsquigarrow \psi$ cannot be applied if there is no world w in which $w \models \varphi \wedge \psi$ or for all such worlds a lock, $lock(w', w)$, exists for some w'.

Algorithm 9.2 Rule application

function APPLY($(\varphi \rightsquigarrow \psi), W, \leq$)
  $max \leftarrow max(\leq)$
  for all $w \in W$ **do**
    if $w \models \varphi \wedge \neg\psi$ **then** $W_c \leftarrow w$
    if $(w \models \varphi \wedge \psi)$ and $\neg\exists w'(w' \in W \wedge (w', w) \in lock)$ **then**
      $o(w) = max + 1$
      $W_s \leftarrow w$
  if $W_s = \emptyset$ **then return** $\bot$
  for all $w \in W_s, w' \in W_c$ **do** $lock(w, w')$
  return $\top$
end function

Proposition 9.8. *Given an initial ordering $\leq$, a set of rules $R = \{r_1, \ldots, r_n\}$ where each r_i is of the form $\varphi_i \rightsquigarrow \psi_i$ and a QDT-model M, the result of successfully applying rules r_1 to r_i, $0 < i \leq n$ is an ordering which respects rules $\{r_1, \ldots, r_i\}$. An ordering $\leq$ respects a rule r_i iff in the most normal (or preferred) world where φ_i is true in that world, ψ_i is also true.*

Proof. When $i = 1$ no previous rules have been applied, so we only have to show that the model respects rule r_1 after successful application. We have $o(w) = 1$ for all worlds w. Applying r_1 can only fail if no worlds entail $\varphi_1 \wedge \psi_1$ or all entailing worlds are locked. Since $lock = \emptyset$ initially, only the former can be the case. But then the rule would describe an impossible world and cannot be applied. Otherwise, after applying r_1, it is entailed by the model, since for all worlds w where $w \models \varphi_1 \wedge \psi_1$ we have $o(w) = 2$ and the o-value of all other worlds is unchanged. Thus the worlds entailing r_1 are most preferred so the rule itself is entailed by the model.

When $i > 1$ we assume that all rules up to and including r_{i-1} have been applied successfully. We therefore have

$$M \models (\varphi_1 \rightsquigarrow \psi_1) \wedge \cdots \wedge (\varphi_{i-1} \rightsquigarrow \psi_{i-1}).$$

Let $l_i = \{(w, w') \mid w \models \varphi_i \wedge \psi_i \text{ and } w' \models \varphi_i \wedge \neg\psi_i\}$ be the set of locks between worlds with contradictory consequents of a rule $\varphi_i \rightsquigarrow \psi_i$. Before applying r_i the set *lock* contains

$$lock = l_1 \cup \cdots \cup l_{i-1}$$

Rule r_i can then be applied if there is at least one world w in which $w \models r_i$ and where w is not the second entry of a pair in *lock* (i.e., there is a world entailing r_i which is not locked by another world). If there is no such world then either the rule describes an impossible world and should be rejected, or a previously applied rule contradicts it, which also means it should be rejected. Otherwise the rule will be successfully applied resulting in a model entailing all rules up to and including r_i:

$$M \models (\varphi_1 \rightsquigarrow \psi_1) \wedge \cdots \wedge (\varphi_i \rightsquigarrow \psi_i),$$

and a new *lock* set: $lock' = lock \cup l_i$. Assuming that the rule is successfully applied we know that for all w in which $w \models r_i$ we have $o(w) = max(\leq) + 1$. Clearly r_i is then entailed by the model. We then have to show that all rules up to r_i are still entailed as well.

Consider rule r_j where $0 < j < i$. Rule r_j was entailed by the model before applying r_i. Therefore there are worlds w_j that are not locked, where $w_j \models \varphi_j \wedge \psi_j$, and w'_j where $w'_j \models \varphi_j \wedge \neg\psi_j$, and for all such worlds we have that $o(w_j) > o(w'_j)$ and $(w_j, w'_j) \in lock$. Thus all worlds contradicting r_j are locked relative to those entailing it. If $w'_j \in W_s$ for some w'_j then some of the sought worlds are locked by r_j, but since W_s only contains unlocked worlds, this cannot be the case. Therefore no worlds w'_j will be given a higher o-value than any w_j world. Furthermore, since w'_j contains all the worlds that could invalidate r_j, clearly r_j is still entailed after applying r_i. □

Even though we can successfully apply a set of rules, the function can be further optimized to maximize the number of successful applications of rules. Note that the use of a locking mechanism decreases the number of worlds that can be moved around every time a rule is successfully applied. Therefore, by minimizing the number of worlds being locked in each iteration, we maximize the number of rules that can be applied. The function $s : R \to \mathcal{N}$ gives each rule a score, where rules with many atoms and operators receive higher scores than rules with few.

$$\begin{aligned}
s(\top \rightsquigarrow \psi) &= s(\psi) - 1 \\
s(\varphi \rightsquigarrow \psi) &= s(\varphi) + s(\psi) \\
s(\varphi \wedge \psi) &= s(\varphi) + s(\psi) + 1 \\
s(\neg\varphi) &= s(\varphi) + 1 \\
s(\top) &= 0 \\
s(p) &= 1
\end{aligned}$$

By applying the highest valued rules (the most specialized) first, we ensure that as few worlds as possible are locked. Notice that rules where the antecedent is $\top$ will be penalized, since they are very general, whereas $\top$ in the consequent is ignored.

The algorithm GENERATE : $(F, C, I, R_P, R_N) \to (W, \leq_P, \leq_N)$ generates orderings for preference and normality, and works as follows (Algorithm 9.3). Retrieve relevant atoms and generate an initial model of possible worlds (based on both sets of rules). Then, for both preference and normality rules, we sort the rules descending according to their s-value using SORT(R), and apply each relevant rule in R using APPLY($\varphi \rightsquigarrow \psi, W, \leq$). Finally, the algorithm returns the possible worlds and the orderings, which respects all successfully applied rules.

Remark (*Application of equally general rules*). The need for constraining the order of rule application touches upon a shortcoming of the model generation; rule application may fail, if previously applied rules have locked the matching worlds. In many cases this is actually a good thing, since it does not make sense to first apply a rule $r_1 = \varphi \rightsquigarrow \psi$ and then later $r_2 = \varphi \rightsquigarrow \neg\psi$. r_1 and r_2 are clearly contradictory rules, and both should not

Algorithm 9.3 Model generation

function GENERATE(F, C, I, R_P, R_N)
  $At \leftarrow$ RETRIEVE_ATOMS($F, C, R_P \cup R_N$)
  $W \leftarrow$ INIT(At, I)
  $\leq_P \leftarrow o(W)$
  $\leq_N \leftarrow o(W)$
  for all $(\varphi \rightsquigarrow \psi) \in$ SORT(R_P) **do**
    APPLY($(\varphi \rightsquigarrow \psi), W, \leq_P$)
  for all $(\varphi \rightsquigarrow \psi) \in$ SORT(R_N) **do**
    APPLY($(\varphi \rightsquigarrow \psi), W, \leq_N$)
  return $(W, \leq_P, \leq_N)$
end function

be applied at once, since we cannot both expect (or prefer) ψ and $\neg\psi$ when φ is true. However, if two rules receive the same score they will be applied in a non-deterministic order which could lead to a situation where applying the rules in one order results in one model, and applying in another order results in a different model. It might even be the case that we can apply both rules using one ordering, while another ordering rejects one of the rules.

Consider rules $R = \{\top \rightsquigarrow A, \top \rightsquigarrow B\}$ and possible worlds $W = 2^{\{A,B\}}$. The rules have equal score and they will therefore be applied in a non-deterministic order. If $\top \rightsquigarrow A$ is applied first the ordering will be $AB < A\overline{B} < \{\overline{A}B, \overline{AB}\}$, whereas the ordering will be $AB < \overline{A}B < \{A\overline{B}, \overline{AB}\}$ if $\top \rightsquigarrow B$ is applied first. The rules satisfy the model in both cases; in the most preferred world(s) both A and B hold, but the ordering of less preferred world differs. We argue that even though this is the case, it is clear that as long as the rules have been successfully applied they are satisfied by the model, which means that the model can be used by the agent to reason about its influences by taking its preferences into account. In situations where certain orderings might reject a rule while other orderings would not, it is evident that the latter ordering is favored[3]. If this is the case, the agent might simply monitor the rule application, and if the algorithm rejects a rule given a certain ordering, the agent can attempt to apply the equally general rules in a different order. However, if all orderings result in rejection of one of the rules, this indicates that some of the rules contradict each other, suggesting that not all the rules can be consistently applied to the model.

9.4 The Electronic Auction House

This section serves to show how the decision procedure can be used by agents in the EAH scenario to make decisions when influenced by AORTA. We consider an agent, Bob,

[3] After all, the aim is to apply as many of the rules as possible.

having the following preferences:

$$\neg badInfo(Agent) \rightsquigarrow bid(bob, lamp)$$
$$bid(bob, lamp) \rightsquigarrow participates(bob, lamp)$$

That is, if he has not provided bad information, he prefers to bid on the lamp, and if he does bid on a lamp, he prefers to be participating in the auction for that lamp.

Furthermore, we consider the following normality rules:

$$bid(bob, lamp) \rightsquigarrow verified(bob)$$
$$badInfo(Agent) \rightsquigarrow \neg verified(bob)$$
$$bid(bob, lamp) \wedge badInfo(Agent) \rightsquigarrow \neg participates(bob, lamp)$$

In the following, when talking about a state, we write $\overline{X}$ to denote $\neg X$ and we abbreviate the predicates as follows: B denotes $bid(bob, lamp)$, I denotes $badInfo(bob)$, P denotes $participates(bob,lamp)$, and V denotes $verified(bob)$.

We can then refer to a state where, e.g., Bob has participated and has provided bad information as follows: $\overline{B}IP\overline{V}$. We thus write conjunctions by writing literals next to each other.

We assume Bob can control (directly or indirectly) the following literals:

$$C = \{B, P, V\}.$$

In this example, Bob can thus (indirectly) control his verification (by contacting the manager), but we assume that he has already registered with bad information, and thus cannot at this point control whether other agents believe that he has provided bad information.

Furthermore, let us assume the following situation is impossible:

$$I = \{\neg P \wedge V\}.$$

That is, Bob cannot be verified if he does not participate.

We can then generate a QDT-model for Bob having to decide whether to bid, i.e., with the following set of influences:

$$F = \{B, \neg B\}.$$

To illustrate how the model generation works, we show the generation of Bob's preference but with V omitted (Figure 9.1). The application of the first rule moves $B\overline{I}P$ and $B\overline{IP}$ to the top of the ordering and locks $\overline{BIP}$ and $\overline{BI}P$, since these contradict the rule. By applying the second rule, we reach the final configuration, where $B\overline{I}P$ and BIP are the most preferred worlds. In other words, the most preferred worlds are those where the agent bids and participates at the same time (regardless of whether he has provided bad information).

The full QDT-model is shown in Figure 9.2. Figure 9.2(a) shows Bob's preferences including V, but the outcome is the same: the most preferred worlds are those where Bob bids and participates at the same time. Figure 9.2(b) shows the expectations from the

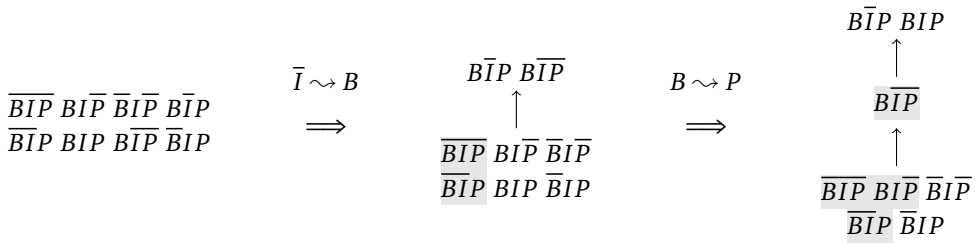
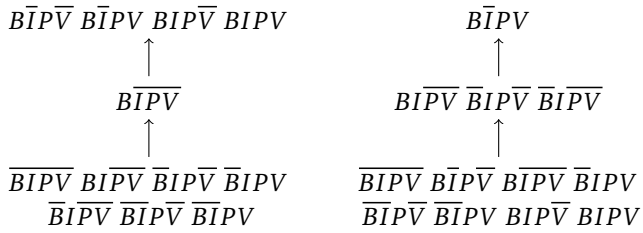


Figure 9.1: Generation of Bob's preferences. Locked worlds are shown with a gray background. Note that we do not show which world has locked it.



(a) Bob's preferences **(b)** Expectation

Figure 9.2: The preference and normality orderings generated using the rules and prohibitions specified by the organization and Bob.

organization. The most normal world is one, where Bob has not provided bad information and is free to bid, participate and be verified. If Bob has provided bad information (second row of worlds), we can expect that he will not become verified.

Bob can make a decision using the model and his influences. Let us consider two situations: one where he has provided bad information and one where he has not. We assume that he initially participates in the auction for the lamp.

Bob provided bad information

We have that $\Sigma = \{I, P\}$. Since $I \notin C$, we can ignore all worlds where $\neg I$ holds (i.e., nothing Bob can do will change the fact that he has provided bad information). From the definition of relative preference, we can see that $B \leq_P \neg B$, since all worlds where $\neg B$ holds are less preferred than some worlds where B holds. Therefore, $Pref = \{B\}$.

To compute which is more tolerated, we first consider the expected consequence of each:

$$EC(B) = \neg P \qquad EC(\neg B) = \neg V$$

Thus, by bidding, Bob can expect to be thrown out of the auction, and by not bidding, he can expect to not become verified. Of course, he might also be thrown out, but this is not more expected than not being thrown out.

We then consider which of the consequences is most tolerated. We can see that not being verified is tolerable (in one of the most preferred worlds, V does not hold), whereas not participating is not (in all of the most preferred worlds, P holds), so $\neg V$ is more tolerable than $\neg P$.

Thus $Tol = \{\neg B\}$ and $Dec = Tol = \{\neg B\}$.

Bob did not provide bad information

We have that $\Sigma = \{\neg I, P\}$, and we can ignore all worlds where I holds. Again we have that $Pref = \{B\}$, since in the most preferred worlds, B holds.

The expected consequences in this situation are:

$$EC(B) = V \qquad EC(\neg B) = \top$$

By bidding, Bob can expect to become verified, and he expects no consequences from not bidding.

Being verified is tolerated since in one of the most preferred worlds, V holds, and having no consequences is tolerated as well. Furthermore, since the agent has no preference regarding verification (both V and $\neg V$ holds in some most preferred world), the consequences are equally tolerable and both influences are therefore tolerable.

We thus have $Tol = \{B, \neg B\}$ and therefore $Dec = Pref \cap Tol = \{B\}$.

Note that Bob was labeled neither "social" nor "selfish" (as would have been the case by using the orderings proposed by Broersen et al. [2001] in the BOID architecture). Both his preference and the expected consequences are taking into account, and this leads to the results above. By choosing to bid, the agent is not strictly social (by complying with the organizational norms). Rather is his decision based on the fact that he (as a result of his preferences) cannot tolerate not following the norms.

If we, e.g., change Bob's preferences to the single rule, $\top \rightsquigarrow B$, then he will bid on the lamp, regardless of whether he has provided bad information, because his preferences tolerate not participating in the auction. In other words, he would rather bid (and risk being thrown out) than not bid, which would mean that he would definitely not buy the lamp.

9.5 Concluding Remarks

We have argued that conflicts are prone to arise when agents interact in open societies and enact roles in an organization, since their own desires may be in conflict with obligations toward other agents or the obligations of the role(s) they are enacting. We have discussed why obligations along with desires should be considered influences on the agent's behavior rather than being seen as desires being imposed onto the agent by other entities. Since

influences do not necessarily represent states the agent wants to achieve, they should only be pursued if the agent can tolerate their consequences.

Our approach to decide which influences to commit to, which is based on qualitative decision theory, is an attempt to let the agent reason about the influences without taking into account that one influence is a desire, and another is an obligation, since such bias can result in labeling the agent "selfish" or "social" in advance. The approach works by including the consequence of bringing about a state in the reasoning, thus letting the agent consider its preferences, without choosing something that results in an intolerable state. We have argued that this indeed lets the agents reach a decision without strictly preferring certain types of influences to others.

We furthermore have developed a simple method that can generate models to be used in the reasoning process by the use of expressions describing the agent's preferences. By use of a simple locking mechanism, the method generates models, which respect non-contradictory rules specified by the agent such that it is possible to make a decision among a set of influences. The simple nature of the method also allows us to generate the models on the fly, so that if the agent's preferences change during execution a new model can be generated. Since the method works by generating all possible states of relevant sub-models, it may prove to be inefficient in cases that are more complex. Even though we only consider sub-models, it would be natural to investigate how to optimize this. Furthermore, since rules of equal generality are applied non-deterministically, different models may emerge, though they satisfy the same sets of rules; although our goal was to create models satisfying rules, we believe a deterministic procedure is desirable and it could be an interesting direction for future work.

CHAPTER 10

AORTA as a Decision Influence

We proposed a consequence-based decision-making procedure in the previous chapter and showed that it enabled agents to decide what to do by also considering the consequences of achieving their objectives.

In this chapter, we further elaborate on this by integrating the approach into an APL, thereby making it possible to consider AORTA a decision influence. In other words, we show that it is (practically) possible for agents to decide whether to follow organizational norms or own desires based on the expectations from the organization and the agent's own preferences.

Since norms model expectations from an organization towards the agents participating in it, it is natural to consider whether this can be used as a way to describe expectations in the QDT-model. We show a clear relation between expectations in this model and norms in the AORTA metamodel (Section 10.1). We furthermore describe a proof of concept implementation of the system in Java, that can be integrated in APLs to perform decision-making based on our notion of decision influences (Section 10.2).

10.1 Decision-Making with AORTA

A cognitive agent with an AORTA-component will be influenced by the reasoning performed by the component. This is evident from the fact that commit(ϕ) adds ϕ to the agent's goal base, which means the agent has to decide whether to complete that goal. Usually, without any additional information, the agent will simply adopt the goal. In this section, we show that consequence-based decision-making can be used to make an informed decision about this.

First, we assume that the agent has declared a number of preferences regarding the system to allow us to generate a model of those preferences. Second, we assume that the agent has a procedure to select a goal to pursue based on the current set of goals. We can then filter the current set of goals by using $\mathscr{M}$ to decide whether a goal should be fulfilled, and pass the filtered set of goals to the agent's goal selection procedure.

The procedure can then be used for decision-making with AORTA in two steps. We first generate a model, $\mathscr{M}$, which is based on the agent's preferences and the norms of the system. Then, we can use the procedure to filter the agent's goals.

Note that while AORTA uses different kinds of reasoning formulas (**org**(ϕ), **bel**(ϕ), etc.), we only make a distinction between agent beliefs and organizational beliefs. There are two reasons for this: first, as mentioned, we do not refer to goals explicitly, but rather the state that the goal *achieves* (i.e., a belief). Second, by considering the agent's prefer-

ences, it seems reasonable to omit organizational options, which are not typically desirable, but rather means to an end.

10.1.1 Norms as expectations

As has been advocated throughout the thesis, norms define expected behavior in an organization. As such, it is not hard to see why using norms for creating the normality ordering in a QDT-model is enticing. We consider the conditional norms of AORTA to specify rules for what we expect happen (i.e., what *normally* happens) in the organization.

We understand a conditional obligation, $O_r(p < \delta \mid c)$, as an obligation to achieve p before δ given c. Another way to understand this is that we *expect* p to be the case when both δ and c hold. We let p,δ and p be reasoning formulas. This leads to the following rule:

$$(c \wedge \delta) \rightsquigarrow p.$$

We also want to capture the fact that, sometimes, norms are violated. In that case, we expect the agents in the organization to detect the violation:

$$(c \wedge \delta \wedge \neg p) \rightsquigarrow \mathbf{org}(\mathsf{viol}(a, r, obliged, p)),$$

where a is the name of the agent violating the obligation. Conversely, for a prohibition, $F_r(p < \delta \mid c)$, we expect that if c holds and δ does not hold, p does not hold either:

$$(c \wedge \neg\delta) \rightsquigarrow \neg p, \qquad (c \wedge \neg\delta \wedge p) \rightsquigarrow \mathbf{org}(\mathsf{viol}(a, r, obliged, p)).$$

By using these rules together with the agent's preferences, we can generate a QDT-model, which the agent can use to decide whether to follow the organizational objectives or its own goals.

Example 10.1 (Generated rules)**.** *Consider the following conditional norms of the EAH:*

$O_{buyer}(\mathbf{bel}(verified(Me)) < \mathbf{bel}(bid(Me, Item)) \mid \mathbf{bel}(registered(Me))$

$O_{manager}(\neg\mathbf{bel}(participates(Ag, Item)) < \top \mid \mathbf{org}(\mathsf{viol}(Ag, buyer, obliged, \mathbf{bel}(verified(Ag)))))$

$F_{manager}(\mathbf{bel}(verified(Agent)) < \bot \mid \mathbf{bel}(badInfo(Agent)) \wedge \mathbf{bel}(registered(Agent)))$

We consider the agent Bob wanting to buy a lamp. We can then translate the rules into the following set of normality rules, with variables replaced by atoms accordingly, since the procedure requires ground predicates:

$\mathbf{bel}(registered(bob)) \wedge \mathbf{bel}(bid(bob, lamp)) \rightsquigarrow \mathbf{bel}(verified(bob))$

$\mathbf{bel}(registered(bob)) \wedge \mathbf{bel}(bid(bob, lamp)) \wedge \neg\mathbf{bel}(verified(bob)) \rightsquigarrow \mathbf{org}(\mathsf{viol}(bob, buyer, obliged, \mathbf{bel}(verified(bob))))$

$\mathbf{org}(\mathsf{viol}(bob, buyer, obliged, \mathbf{bel}(verified(bob)))) \rightsquigarrow \neg\mathbf{bel}(participates(bob, lamp))$

$\mathbf{org}(\mathsf{viol}(bob, buyer, obliged, \mathbf{bel}(verified(bob)))) \wedge \mathbf{bel}(participates(bob, lamp))$

$$\rightsquigarrow \mathbf{org}(\mathsf{viol}(mike, manager, obliged, \neg\mathbf{bel}(participates(bob, lamp))))$$

$$\mathbf{bel}(badInfo(bob)) \wedge \mathbf{bel}(registered(bob)) \rightsquigarrow \neg\mathbf{bel}(verified(bob))$$

$$\mathbf{bel}(badInfo(bob)) \wedge \mathbf{bel}(registered(bob)) \wedge \neg\mathbf{bel}(verified(bob)) \rightsquigarrow \mathbf{org}(\mathsf{viol}(mike, manager, forbidden, \mathbf{bel}(verified(bob))))$$

The first rule states that if Bob has registered and has bid on the lamp, we would expect him to have been verified. The second rule states that if this is not the case, we expect the organization to detect this violation.

10.1.2 Making a choice

Having a number of normality-rules using the translation above, we can use the decision procedure on the agent's influences to decide what to do. Rather than just inputting all the agent's influences, however, we process the agent's goals one at a time. That is, for each influence, g, we generate a model to check if g is more preferred than $\neg g$ (and vice versa). If $g \in Dec$, then the agent selects the goal and proceeds to try to achieve it, otherwise the goal will not be selected.

This makes the decision procedure tailored to the individual influence, since only the relevant rules will be applicable. Furthermore, this effectively filters out any undesirable and intolerable goals, but still allows the cognitive agent to perform its goal selection on the remaining set of goals.

10.2 Practical Decision-Making

To better experiment with the procedure and show its usefulness, we have created a proof of concept implementation in Java[1]. We have already argued for the choice of Java for AORTA in Chapter 6, and it only makes sense to make sure that the decision procedure can be used in connection with various APLs and AORTA.

The implementation faithfully implements the semantics of the QDT-formalism, and is as such not especially efficient. However, as a proof of concept, it fully serves its purpose, as it enables agents in APLs (*Jason*, in this case) to make decisions based on expected consequences.

The model generator includes a small language for inputting rules, such that we can easily experiment with different configurations. Table 10.1 shows an example of a set of rules based on the conditional norms of the AORTA metamodel.

Once generated, the decision model, $\mathscr{M}$, implemented in a `DecisionModel` class provides methods for getting the most preferred influences, the most tolerable consequences, and the set of decisions. All of these faithfully implement their counterpart, as described in the previous chapter.

[1]The implementation is available at `https://github.com/andreasschmidtjensen/decisionmaking`.

Table 10.1: Bob's preferences and the expectations from the organization as generated from the conditional norms of the metamodel. For simplicity, **bel** and **org** has been omitted from the predicates.

```
Preferences:
  bid(bob,lamp) => participates(bob,lamp).
  participates(bob,lamp) => ~viol(bob,buyer,obliged,verified(bob)).

Expectations:
  registered(bob) && bid(bob,lamp) => verified(bob).
  ~verified(bob) && registered(bob) && bid(bob,lamp) => viol(bob,buyer,
      obliged,verified(bob)).

  viol(bob,buyer,obliged,verified(bob)) => ~participates(bob,lamp).
  participates(bob,lamp) && viol(bob,buyer,obliged,verified(bob)) =>
      viol(mike,manager,obliged,~participates(bob,lamp)).

  badInfo(bob) && registered(bob) => ~verified(bob).
  badInfo(bob) && registered(bob) && verified(bob) => viol(mike,manager,
      forbidden,verified(bob)).

Impossible states:
  ~registered(bob) && verified(bob).
  ~registered(bob) && participates(bob,lamp).
  ~registered(bob) && bid(bob,lamp).
```

10.2.1 Decision-making in *Jason*

Jason, being highly extensible, is an obvious choice for implementing such procedure. In order to use the procedure for an agent, we must specify the preference and expectation rules, and the set of controllable atoms. This is done in the project file as follows:

```
MAS project {
    infrastructure: Centralised
    agents:
        bob [dm="eah.dm",c="bid(bob,lamp) && registered(bob) &&
            participates(bob,lamp) && verified(bob) && viol(bob,buyer,
            obliged,verified(bob))"] agentClass dm.DMAgent;
}
```

Here, `dm` specifies the location of the file containing the preference and normality rules (Table 10.1), and `c` is the conjunction of controllable atoms.

The decision procedure is implemented in the agent class, `DMAgent`, by extending the agent's *event selection function*. Events in *Jason* include perceived changes in the environment and changes to the agent's own goals. In each reasoning cycle, the agent selects one event to process. The standard selection function simply picks the first element from the queue of events.

Our implementation takes the first event, and if it is an achievement goal, `+!`g, it executes the decision procedure for $F = \{g, \neg g\}$. Then, if $g \in Dec$, the event is selected,

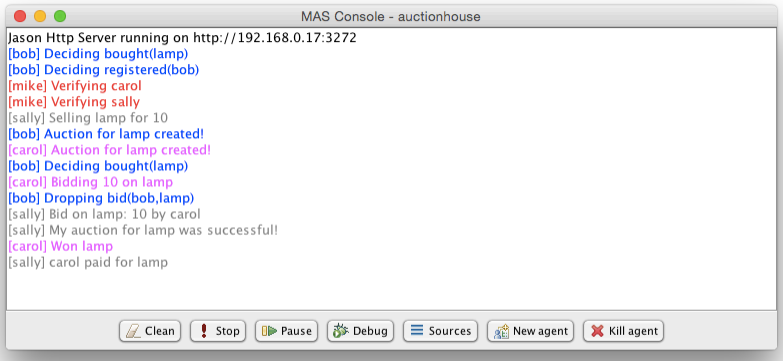


Figure 10.2: Excerpt from the *Jason* execution of the EAH with consequence-based decision-making.

and otherwise it is dropped. This continues until either an event has been selected or there are no events left to consider.

By processing the goals as events, we effectively filter undesirable goals as soon as they are added to the agent, since the addition of a goal adds the goal as an event. Thus, if a goal is not tolerated, the agent will not intend to complete it.

10.2.2 Decision-making in AORTA

Since AORTA merely acts as a decision influence, decision-making with AORTA requires the component to be connected to a cognitive agent. By using our integration of AORTA with *Jason*, we can use the implementation of the `DMAgent` to let commitment of goals from AORTA act as an influence for the agent in *Jason*.

When executing the commit(ϕ) action in AORTA, a goal event is added to *Jason*, and our implementation of the event selection function then decides whether this specific goal can be executed based on the agent's preferences and the expectations from the organization.

Figure 10.2 shows an excerpt from an execution of the agents in the EAH scenario in *Jason*. Each agent prints what it is doing: Mike verifies the agents, Sally puts the lamp up for sale, and so on. Notice the output from Bob:

```
[bob] Deciding bought(lamp)
[bob] Deciding registered(bob)
[bob] Deciding bought(lamp)
[bob] Dropping bid(bob,lamp)
```

Each line indicates the result of the decision-making process. In each case, a model of the system has been generated based on the influence and the rules in Table 10.1. In the first three lines, Bob decides to commit to the goals, since his preferences and the expectations allow it (he has no explicit preference of whether to register or buy the lamp). He thus commits to buy the lamp[2] and to register. In the last line, he decides to drop the bid-goal, since – as we saw in the example in the previous chapter – he does not want to be thrown out.

If we change his preferences to the single rule `true => bid(bob,lamp)`, he decides to bid, since he now tolerates the risk of being thrown out of the auction. Thus, by specifying the agent's preferences, we have shown that the agent can base its decisions on the consequences of violating the organizational norms and only complete tolerated objectives.

10.3 Concluding Remarks

We have shown that our formalism for consequence-based decision-making can be implemented and applied to agents in an actual APL. By integrating the implementation with *Jason*, agents are able to decide whether to adopt goals when they are added from various influences, including their own plans, commit-actions in AORTA, and via requests from other agents.

The preference and expectation rules are specified in a simple language that makes it possible to easily experiment with different configurations, to, e.g., investigate how the agent reacts when its preferences change.

Finally, we showed a clear relation between the conditional norms of the AORTA metamodel and expectations in the decision procedure. By using this relation, we showed that it was possible for agents to reason about the consequences of violating the norms of an organization.

[2] He decides to buy the lamp twice because of the way *Jason* executes plans: a goal is considered done once the plan is finished, so the AORTA-component commits to the goal multiple times, since it (when perceived as an achievement goal) is not yet completed.

CHAPTER 11

Organization-Aware Agents

Through the four parts of this thesis, we have presented our approach to organizational reasoning. In this chapter, we claim that by using the AORTA reasoning component, agents do in fact become organization-aware.

We listed a number of requirements for organizational reasoning in Chapter 3. We go through each of the requirements below and show how AORTA enables that kind of organizational reasoning. We do not say that our approach fulfills *every* requirement, but we argue that the requirements we *do* fulfill provide sufficient evidence for our claim.

In Section 11.1, we discuss the reasoning requirements for entering an organization. We address the requirements for participating in an organization in Section 11.2 and finally, we discuss the requirements for leaving an organization in Section 11.3.

11.1 Entering an Organization

Which role(s) can the agent successfully play? We assume that by having the capability to fulfill the objectives of a role, the agent can successfully play that role. In AORTA, the agent primarily enacts roles based on options generated in the OG-phase. A role is considered an option if the agent has the capability to fulfill at least one of the role's objectives. In this way, success is partially is measured automatically by means of the agent's capabilities. However, it is possible to refine enactment further in a reasoning rule that, e.g., require that more objectives can be fulfilled by the agent:

```
1 role(R) : org(role(R,Os)), ~(bel(member(bel(O),Os)), ~cap(O))
2         => enact(R).
3 role(R) : org(role(R,Os)), bel(member(bel(O1),Os), member(bel(O2),Os),
4             O1 \= O2), cap(O1), cap(O2)
5         => enact(R).
```

The first example requires that the agent is capable of fulfilling all objectives of the role; the second requires that at least two objectives can be fulfilled. Of course, capability does not guarantee fulfillment, but it provides a reasonable basis for assuming that the objectives can be fulfilled.

What power relations will be put in place when entering? Power relations in AORTA are defined by the dependency relation, $\mathsf{dep}(r_1, r_2, o)$. The intuitive meaning of the relation is that r_1 *depends on* r_2 for the completion of o. This is, in some sense, a backwards power relation. By enacting r_1, the agent will need agents enacting r_2 in order to complete o. For example, the customer depends on the manager for the verification process in the EAH scenario.

Carabelea et al. [2005] define power relations due to authority as follows

$$\text{power\_over}(A, B, o) \equiv \exists r_1, r_2 : \text{plays}(A, r_1) \wedge \text{plays}(B, r_2) \wedge \text{authority\_over}(r_1, r_2, o).$$

That is, agent A as power over agent B with regards to o, if A enacts a role that has authority over a role played by B with regards to o. Compare this to the precondition of the Delegate transition rule:

$$MS \models \mathbf{org}(\text{dep}(r_1, r_2, o)) \wedge \mathbf{org}(\text{rea}(A, r_1)) \wedge \mathbf{opt}(\text{obj}(o)).$$

That is, if there is a dependency relation between r_1 and r_2, the agent (A) enacts r_1 and o is an objective, it can be delegated to agents enacting r_2. Agents enacting r_2 then have power over agent A, since he depends on them for the completion of o.

In terms of reasoning about entering the organization, it is possible for the agent to decide to, e.g., only join if it trusts the agents potentially having power over it:

```
1 role(R) : org(dep(R,R2,O)), ~(org(rea(A,R2)), ~bel(trusts(A)))
2         => enact(R).
```

Another, more sophisticated power relation is related to violation and sanctioning. If the violation of a norm activates a contrary-to-duty norm for another agent, that agent has power over the violating agent. For example, if the manager has an obligation to ban agents that violate the verification-obligation, agents that consider violating this obligation can take this into account when deciding whether to join. For example, if the agent trusts the other agent, it may believe that it will not be sanctioned and choose to enact the role:

```
1 role(R) : org(cond(R,obliged,O,D,C),
2             org(cond(R2,obliged,_,_, org(viol(A,R,obliged,O)) ))),
3             ~(org(rea(A,R2)), ~bel(trusts(A)))
4         => enact(R).
```

What capabilities are required? AORTA does not as such specify requirements for enacting a role. However, the agent *can* reason about the role's objectives and compare this to its own capabilities, as discussed above.

What capabilities are gained? The AORTA-component does not provide agents with new capabilities for acting in the environment. It, of course, provides agents with the ability to enact roles and commit to objectives, but these are not bound to the enactment of specific roles.

Why should the agent enter the organization? Given the discussion above, can the agent reason about why it should enter the organization? We firmly believe that the ability to reason about which roles to play and the power relations in the organization provides the agent with a strong foundation for deciding whether to join.

For example, the agent can reason that it can successfully play a role having objectives that coincide with the agent's goals, and furthermore that it will depend on other agents for the completion of some of these objectives. Then, if it trusts these agents, it can choose to enter the organization, since this will aid the agent in the completion of its personal goals.

11.2 Playing Roles in an Organization

What is required to complete a given objective? We consider an objective as a partial state description. That is, an objective *verified*(*bob*) is an objective to be in a state, where *verified*(*bob*) holds. As per the OCMAS principles proposed by Ferber et al. [2003], the AORTA metamodel does not state how to complete an objective. However, the agent can still use the organizational structure to reason about the requirements for fulfilling an objective. This can then be used to decide how to act on the objectives that have been generated in the OG-phase.

If the agent has the capability to complete an objective, it can simply decide to commit to it:

```
1 obj(O) : cap(O) => commit(O).
```

If the objective is composed of several sub-objectives, this relation can be used to, e.g., complete all the sub-objectives first:

```
1 obj(O) : org(obj(O,Subs)), bel(member(S,Subs)), ~bel(S) => commit(S).
2 obj(O) : org(obj(O,Subs)), ~(bel(member(S,Subs)), ~bel(S)) => commit(O).
```

That is, if there is a sub-objective that has not yet been fulfilled, the agent should commit to it. When all sub-objectives have been fulfilled, the agent commits to the main objective.

How can the agent complete an objective without having the required capabilities? If the agent, based on the reasoning above, cannot complete an objective because it does not have the required capabilities, it can delegate the objective to an agent that can (or should be able to). This requires that the organization is *efficient* (cf. Chapter 8):

```
1 obj(O) : ~cap(O), opt(send(R, achieve, O)), org(rea(A,R))
2          => send(A, goal(O)).
```

This does not ensure that the objective is eventually fulfilled, since the other agent may choose not to fulfill the objective. However, proper sanctioning mechanisms can be put in place to accommodate this.

How can the agents detect other agent's violations? Detection of violation happens automatically in the NC-phase (using the Obl-Viol or Pro-Viol transition rules). Since the component is a part of the agent, this requires that the agent somehow is made aware that a violating state has been reached by another agent (e.g., by perceiving that the deadline has been reached).

What are the consequences of violating a norm? The immediate consequence of violating a norm is the execution of the NC-phase, which adds a viol-predicate to the organizational knowledge base. The real question is then: how can the agent deduce the consequences of the viol-predicate?

The organization itself is not an "active" entity, and will not, e.g., automatically remove violating agents. As we have argued, and as it is usually the case in real organizations, the participating agents are responsible for sanctioning violating agents. There are multiple ways of sanctioning: one is simply to ignore agents that have violated some obligations. However, as the agents usually consider each other as black boxes, this is hard to detect. We thus consider the *visible* consequences, which in our case can be one of two things:

1. Contrary-to-duty norms.
2. Consequence-based decision-making.

Contrary-to-duty norms can, as discussed in the previous section be reasoned about as part of a reasoning rule. This makes it possible to reason that, e.g., if the agent violates the norm, another norm is activated. The agent then has to reason about whether this is acceptable or not:

```
1 true : bel(badInfo(Me), want(Item)),
2          org(norm(Me,R,obliged,bel(verified(Me)), bel(bid(Me,Item)))),
3          ~org(cond(_,_,_,_,org(viol(Me,R,obliged,bel(verified(Me))))))
4     => commit(bid(Me,Item)).
```

Here, the agent commits to bidding *only* if there is not conditional norm that is activated if the agent violates the verification-norm.

The second method involves our model for decision-making. Though it is not directly a part of AORTA, we have shown how AORTA can be used as an influence in the model. In this case, the agent can use the model to reason whether the consequences of its decisions are acceptable. It is thus added as a filter *after* AORTA (or another influence) has added a goal to the agent's goal base, but we argue that it still enables the agent to reason about the consequences of, e.g., violation of norms.

11.3 Leaving an Organization

When has the agent fulfilled its duties towards the organization? To answer this question, we must first define the agent's duties towards the organization. Can we define it simply as the objectives of the role(s) the agent enacts? Does it include the responsibility the agent has towards other agents in the organization?

If the former is the case, then the solution is quite simple. The Deact transition rule generates an option to deact a role once every objective has been fulfilled. The agent then simply acts upon this option:

```
1 ~role(R) : true => deact(R).
```

The latter case is more complex. It requires the agent to reason about, e.g., dependency relations, norms and other agents' role enactment. As we have already shown, all of this is possible using AORTA reasoning rules. For example, the agent can reason that all dependencies have been fulfilled:

```
1 ~role(R) : ~(org(dep(R,R2,O)), ~bel(O)) => deact(R).
```

Here, the agent deacts the role if all its own objectives have been fulfilled and no role has power over this agent concerning some unfulfilled objective.

What are the benefits of staying compared to leaving? The perhaps most compelling reason to stay in an organization once ones duties have been fulfilled is the capabilities gained though role enactment. While AORTA does not support this, we argue that it can still be beneficial to stay in an organization even though the agent's objectives have been fulfilled. For example, power relations might be used to fulfill personal goals. Furthermore, by staying in the organization, the agent is able to cooperate with other agents in the organization. For example, the agent might have personal goals that do not directly coincide with the organization's objectives, but will be more efficiently achieved by cooperating with other participating agents.

For example, the agent can reason that it has a personal goal, which is also a subobjective for some objective of another role-enacting agent:

```
1 true : org(obj(O,Subs)), goal(member(S,Subs), S),
2          org(role(R,Os), member(O,Os), rea(A,R))
3     => send(A, goal(S)).
```

This is an attempt to let the other agent achieve the subgoal. While it may be overly optimistic to assume the other agent will fulfill the goal just like that, it shows that such reasoning is indeed possible.

However, as it is not possible to reason about losing capabilities, there are not many reasons for staying in an organization in which ones duties have been fulfilled, so for more useful reasoning about the benefits of staying, we believe that the possibility of acquiring capabilities is necessary.

Can agents that perform repeated violations be removed? As we have argued, violation detection is part of the NC-phase. However, since the agents cannot currently control each other's role enactment, it is not possible to remove other agents (i.e. perform the deact action on their behalf).

Instead, the agents can make it difficult for the agents that repeatedly violate the norms of the organization to participate in the organization. For example, the agent might choose to not cooperate with these agents:

```
1 send(R,tell,X) : org(rea(A,R)), ~org(viol(A,R,_,_)) => send(A,bel(X)).
```

In this example, the agent has an option to *tell* agents playing a specific role something, but effectively ignores agents that have violated an obligation while playing that role.

11.4 Concluding Remarks

We believe our reasoning framework successfully captures the most relevant and required concepts concerning reasoning about entering and participating in an organization. However, as the framework does not support the acquisition of new capabilities following role-enactment, certain aspects are not possible to reason about. This is particularly clear in the deactment-phase, where the agent mostly is able to reason about whether its own duties have been fulfilled. We showed that it is possible to take other agent's violations into account as well to, e.g., make it difficult for violating agents to participate, but it is currently not possible to forcefully remove those agents from the organization.

CHAPTER 12

Conclusion

In this thesis, we have investigated organizational reasoning from the agent's perspective. We have designed and built the AORTA reasoning framework for organization-aware agents and have shown how it can be used to add organizational reasoning capabilities to agents in various agent programming languages. The motivation for this research came from a need to let agents successfully participate in *open systems* where certain rules and norms govern the agents and their behavior.

Our main research goal has been to create a bridge between organizational models and agent programming languages that allows the agents to operationalize the model. We are furthermore convinced that a *general* solution, which allows agents built in different programming languages to reason about different metamodels, is most well suited for open systems. This follows from the fact that open systems inherently invites agents of different nature to participate and a general solution would enable this.

We claim that our framework is a contribution to state of the art for the following reasons:

- It is founded in formal logic and has operational semantics (Part II).
- It has been integrated with existing agent programming languages such that the organizational programming needs to be done only once (Part III).
- It includes a procedure for deciding between organizational and personal influences (Part IV)
- It enables organizational reasoning (Chapter 11).

In the remainder of this final chapter of the thesis, we discuss the results of the project and our main contributions in Section 12.1 and we provide pointers for future research in Section 12.2. Finally, we discuss some visions for organization-aware agents in Section 12.3.

12.1 Main Contributions

We set out to create a bridge between organizational models and agent programming languages. By investigating state of the art (Part I), we identified a number of sub-goals to contribute to the fulfillment of main our research goal. First, a way to specify and operationalize organizational models in a way that captures the main concepts of existing models (Part II). Second, an organization-oriented programming language and a reasoning component for implementing organization-aware agents (Part III). Third, to ensure the agents are still autonomous, a way to decide between own desires and organizational goals (Part IV).

12.1.1 Part II: Adding Organizational Reasoning to Agents

Our goal to create a *general* approach required us to devise a general way of describing organizations. Our solution was the AORTA metamodel, which covers aspects of various existing models, such as roles, objectives and norms. We decided not to include the notion of groups in the model, which means that certain aspects of existing models are hard to capture. For example, the notion of role cardinality in a group in MOISE$^+$ cannot directly be captured by the metamodel. While we can accommodate this by simulating groups (by creating an organization for each group), this does not precisely capture the intention, so the addition of groups in the model could be beneficial. However, we showed the inherent usefulness of the metamodel by providing translations of the OperA model and the MOISE$^+$ model to the metamodel. Furthermore, simplicity has been a very important factor in designing the metamodel. First, a simple model is easy to understand by a programmer, who can get quickly acquainted with the concepts and apply reasoning rules to act on them. Second, by not adding too many concepts at once, we get a better understanding of the concepts we *do* investigate.

We proposed the AORTA organizational reasoning component as a way to operationalize the metamodel. It is characterized by being completely decoupled from the cognitive agent, by its automated reasoning about norms (NC-phase) and organizational options (OG-phase), and by the reasoning rules specified by the designer to act upon norms and options (AE-phase).

Furthermore, we specified the component using structural operational semantics providing us with a formal, rigid description of each part of the component and its behavior during execution. This enabled us to precisely specify each of the phases (using transition rules), and it makes the implementation of the system quite straightforward.

12.1.2 Part III: Engineering Organization-Aware Agents

In the third part of the thesis, we presented our implementation of the AORTA framework. Based on the operational semantics, we implemented the AORTA architecture; an open source Java-based component, designed to be easily integrated with existing agent programming languages. By letting the implementation follow the operational semantics very closely, we get a system that is easy to follow, and its behavior can more easily be investigated (using, e.g., verification).

By integrating the component with several existing platforms (*Jason*, 2APL and the agent infrastructure layer (AIL)), we have shown that our approach indeed is viable. Furthermore, it shows – in combination with the metamodel – that our approach is general enough to integrate various languages with different models.

We implemented the EAH scenario for each of the integrations in which the AORTA metamodel and reasoning rules could be reused as-is. In other words, the component is truly decoupled and independent from the specific agent platform. This advocates its use in open systems, where agents from different kinds of platforms may interact in the same organization, thus needing a way to reason about their organizational coordination.

Finally, we took a step towards actual verification of organization-aware agents. Others have investigated model checking of organizations, but the strength of our approach is that we integrate AORTA with AIL. This enables verification of AORTA integrated with agents built using different programming languages without any extra work (as long as those languages have been implemented using AIL). The greatest drawback of the approach is that it is slow: in order to verify rather simple properties of the system, we had to reduce the number of agents and remove parts of the scenario.

12.1.3 Part IV: Deciding Between Influences

Parts II and III focused on organizational reasoning from a theoretical and practical perspective, respectively. One of the most important criteria for making agents organization-aware is that the agents must still be *autonomous*. This touches upon a potential issue with the framework, since the possibility to commit to organizational objectives can affect the agent's autonomy. Depending on how the integration between AORTA and the cognitive agent has been done, adding an objective to the cognitive agent may force the agent to try to fulfill that objective.

In the final part of the thesis, we therefore proposed a model to solve this problem by adding a filter to the agent's decision procedure that takes consequences of fulfilling a goal into account before deciding to commit to it. By considering both the agent's preferences and the expected outcome of fulfilling the goal, we showed that it was possible for the agents to make qualified context-dependent decisions.

In particular, we showed its usefulness when combined with organizational norms. We added conditional norms of the AORTA metamodel to the decision model and showed how agents could reason about the consequences of violating a norm before deciding whether to commit to certain objectives.

Finally, we devised a simple prototype implementation in Java that could be integrated with *Jason*. This enabled the agent to make decisions based on its preferences, while considering organizational constraints as well.

By adding together the possibility to reason about and act upon an organizational model (Part III) and deciding whether to adhere to the organizational expectations (Part IV), we thus claim that our agents are organization-aware. We showed in Chapter 11 that our component enables agents to perform most of the reasoning requirements we identified in Part I. Since most of the limitations were due to the lack of possibility to enrich the agents with new capabilities upon enacting a role (e.g. access to certain resources in the environment), we are convinced that the AORTA component indeed enables organizational reasoning.

12.2 Future Research Directions

In this section, we discuss some ideas for future research in relation with the framework. While our framework is complete (in the sense that it fulfills our objectives), there are of course things that can be improved or extended. For example, we believe the decision

procedure in Part IV can become more efficient (both in terms of model generation and the implementation in general). Another issue is that of verification: in its current state, it is quite inefficient and to make it more useful, optimizations are required. However, in the following, we focus mainly on the larger picture: how can we include other concepts that can further enrich the framework and will contribute to state of the art.

Our ideas concern centralizing the organization (e.g., by using artifacts), truly open systems in which agents from different languages can use AORTA to participate and, finally, a richer metamodel that can capture more organizational concepts.

12.2.1 Centralized organization

The AORTA component is added to the individual agent to provide it with organizational reasoning capabilities. It keeps a local view of the organization (as an organizational knowledge base), which the agent can reason about and alter using reasoning rules. In the current state of the framework, there is no global view of the organization; when the agents change the organization locally (e.g., by enacting a role), other agents will only know if the agent in question tells them. We have shown that given the AORTA reasoning rules, the agents can synchronize their organizational knowledge, but when it comes to larger, more complex systems, this approach may be inefficient.

We briefly discussed the idea of organizational artifacts in Chapter 3. Most notably, the ORA4MAS approach [Hübner et al., 2010] allows the encapsulation of different organizational models in artifacts that agents can interact with. It would be very interesting to extend our framework to allow the interaction with organizational artifacts. For example, role enactment could be done by executing operations on the artifact, and other agents would be aware of this by sensing properties of the artifact.

12.2.2 Truly open systems

We have advocated for the use of our framework in open systems, since in such systems, agents of different origin can interact with each other. In this thesis, we have shown that agents programmed in different languages can use AORTA, but we have not attempted to let them interact in an open system. We believe, however, that if agents of different origin (using AORTA) enter the same system, they will be able, with little work, to use AORTA to cooperate. For example, the environment interface standard (EIS) [Behrens et al., 2011] provides an interface for connecting agent platforms to an environment.

We find that an investigation of using AORTA in EIS-enabled environments would make an interesting contribution, since it would allow an implementation of open systems in which agents on different platforms can use organizational reasoning to cooperate.

12.2.3 Richer metamodel

The AORTA metamodel provides a generic way to describe concepts such as roles, objectives and norms. We have shown that parts of the OperA and MOISE$^+$ models can be translated to our metamodel, making the use of existing specifications possible.

As discussed in Chapter 11, the metamodel (and the AORTA component) does not provide role-enacting agents with new capabilities. We understand the organization as a collection of concepts that describe the expectations of the agents in the environment. As such, it therefore makes sense to let the agents acquire new capabilities when enacting roles in the environment. For example, by enacting the role of a manager, the agent acquires the capability to verify other agents. In the current state of the system, the capability is always there; the agent only uses the organization as a help for deciding when to exercise the capability.

Taking this shortcoming together with the fact that only parts of existing organizational models can be effectively translated to the metamodel, the investigation of a richer metamodel would clearly benefit the framework.

12.3 Visions

Our work on the AORTA framework was based on the observation that, in some cases, the actions of intelligent agents needs to be restricted. In particular, we have argued that in open systems, restrictions are necessary in order to guarantee (or at least make it plausible) that the boundaries of the system are respected, and that the agents entering the system fulfill their roles. We have provided an example of such open system – the electronic auction house – and have shown the apparent usefulness of organization-awareness by illustrating how agents (using AORTA) are able to base their decisions on both their own desires and the organizational rules of the system.

We conclude the thesis by putting a perspective on our view of organization-aware agents (not necessarily restricted to AORTA agents). We discuss some of the, in our opinion, most promising uses of organization-aware agents.

12.3.1 Open systems

As we have already argued, open systems is an area that will benefit from organization-aware agents. We showed the usefulness in the context of an electronic auction house, but we believe that other kinds of open systems will benefit as well:

Request for Tender. The *request for tender* (RFT) process is a formal, structured process in which (qualified) suppliers are invited to submit bids for construction or for supply of products or services. For example, a company or government may put a building project "out to tender"; that is, publish an invitation for other parties to make a proposal for the building's construction. The stakeholders of the process include the contracting authority, bidders consortium (possibly consisting of several partners), evaluators, and a publication body.

We showed in [Jensen et al., 2015a] that the RFT process can be modeled using organization-aware agents. The process can be viewed as a variant of the auction house process, and it thus makes sense to view it as an open system.

Conference Management System. Conference management systems are widely used as examples for modeling organizations [Dignum, 2004; Tinnemeier, 2011; Zam-

bonelli et al., 2005], since they encompass many interesting situations. Setting up a conference is a multi-phase process, which involves authors, program chair and committee, and reviewers. Such a system is clearly an open system: authors are free to submit their articles to the system, thereby enacting the author role that imposes certain rules upon them. Organization-aware agents can reason about these requirements to ensure that their article complies with the rules of the conference, to submit the paper within the deadline and so on.

The Internet. The Internet is a truly open system: anyone (with internet access) can enter an address in a browser and visit a web-page. Consider, e.g., an intelligent agent acting as a personal assistant with the ability to retrieve information from web-pages. In this case, it would be reasonable to define certain rules of behavior for the agent, such as respecting the `robots.txt` file, which describes pages that search-bots are not allowed to visit, or not visit the same page too often (a group of rogue agents could in principle create a denial-of-service attack). Organization-aware agents could reason about these rules and decide whether to follow them in different situations (it could, e.g., choose to visit a specific page quite often if it believes the content of that page is very dynamic).

12.3.2 Intelligent agents in games

An area that we believe could benefit greatly from intelligent agents is that of *games*. For example, the annual multi-agent programming contest lets participants implement agents that play against each other in complex games [Ahlbrecht et al., 2013].

Another type of game is the real-time strategy (RTS) genre, in which players make decisions and control several units *simultaneously*. RTS games require players to make decisions under time pressure and they should be able to control multiple units at once in order to be successful. A human player may not be able to keep up with all the decisions that have to be made, and is forced to act on a more macroscopic level, controlling large groups of units at the same time rather than individuals. By controlling each individual unit using a dedicated intelligent agent, micromanagement becomes quite natural, since the agents are able to react to local changes.

We have worked on implementing a bridge between multi-agent systems and RTS games in collaboration with a number of MSc students [Frøsig and Andersen, 2015; Kaysø-Rørdam, 2014]. We have successfully implemented a bridge between EIS-enabled multi-agent systems and StarCraft: Brood War [Jensen et al., 2015b], and have furthermore been working on adding organization-awareness to those agents. We believe that by imposing certain high-level rules upon the agents, they are able to better coordinate their tasks.

12.3.3 Agent-based social simulation

Agent-based social simulation (ABSS) is a research area that constitutes the intersection of agent-based computing, social sciences and computer simulation [Davidsson, 2002]. ABSS can be used to simulate certain situations that involve humans to deduce what

might happen in real life in such situation. We will not go into too much detail, but note that the main applications of ABSS are social robots, virtual markets, education and training, and terrorism [Li et al., 2008].

We have worked on simulating theatrical performances using multi-agent systems in collaboration with an MSc student [Spurkeland, 2013]. In this work, we simulated a certain kind of theater that relies heavily on improvisation, using few guidelines to tell the overall story. We have used the OperA model to describe what was expected of the actors, and using that, we were able to formalize a theatrical performance using organizations [Jensen et al., 2013b]. We believe that by using a framework such as AORTA, we enable agents to reason about the formalization and in that way enable them to participate in the simulation. This naturally extends to other kinds of social simulations: by specifying certain boundaries for the agents, we are able to simulate how humans might interact with an organization.

12.4 Final Remarks

In this thesis, we presented the AORTA reasoning framework for organization-aware agents. Our main research goal was to create a framework that effectively creates a bridge between various agent programming languages and organizational models, providing the agents with means to operationalize the model. We strongly believe that we have fulfilled this goal. We do not claim to have built a framework that solves every problem concerning organizational reasoning, but we have shown that agents can use AORTA to become organization-aware, and furthermore, that our approach is viable for different agent programming languages.

APPENDIX A

The Electronic Auction House Scenario

This appendix provides the reader with a full implementation of the EAH scenario that is used throughout the thesis. While parts of the implementation are used in the main part of the thesis to exemplify AORTA, we provide the implementation here in its entirety for completeness.

In the following sections, we provide the implementation of the AORTA metamodel (Appendix A.1), the AORTA reasoning rules (Appendix A.2) and simple implementations of cognitive agents that use AORTA: *Jason* (Appendix A.3), 2APL (Appendix A.4) and AIL (Appendix A.5).

A.1 Metamodel

```
1  ROLES:
2  customer: registered(Agent); bought(Item); sold(Item); paid(Item).
3  buyer: bid(Agent, Item); verified(Agent).
4  seller: auction(Agent, Item); verified(Agent).
5  manager: verified(Agent).
6
7  OBJECTIVES:
8  registered(Agent).
9  verified(Agent): registered(Agent).
10 bought(Item): bid(Agent, Item); paid(Item).
11 sold(Item): auction(Agent, Item).
12 paid(Item).
13 bid(Agent, Item).
14 auction(Agent, Item).
15
16 DEPENDENCIES:
17 customer > manager: verified(Agent).
18
19 NORMS:
20 customer=Me [obliged]: registered(Me) < verified(Me) | agent(Me).
21 customer=Me [obliged]: paidFor(Item) < \+ participates(Me,Item) | won(Me
       , Item).
22 buyer=Me [obliged]: verified(Me) < bid(Me, Item) | registered(Me).
23 seller=Me [obliged]: verified(Me) < auction(Me, Item) | registered(Me).
24 manager [forbidden]: verified(Ag) < false | badInfo(Ag), registered(Ag).
25
26 RULES:
27 registered(Agent) :- registered(Agent, Address, Account).
28 auction(Agent,Item) :- auction(_Id, Item, Agent, _StartPrice, _EndTime).
29 bid(Agent,Item) :- bid(Id, Agent, _B), auction(Id, Item, _A, _SP, _ET).
```

```
won(Agent,Item) :- auction_done(Id,Agent,_B), auction(Id, Item, _A, _SP,
    _ET).
participates(Agent,Item)
  :- auction(Id, Item, _Ag, _SP, _ET), participant(Id, Agent).
paidFor(Item) :- paid(Id), auction(Id, Item, _Ag, _SP, _ET).
```

A.2 Organizational Reasoning

The numbering of the rules (in the comments) refer to Example 5.37 in Chapter 5.

```
if bel(me(Me)) {
  % ROLE ENACTMENT
  %% AR1
  role(customer) : true => enact(customer).
  %% AR2
  role(buyer) : goal(bought(Item)), bel(registered(Me)) => enact(buyer).
  %% AR3
  role(seller) : goal(sold(Item)), bel(registered(Me)) => enact(seller).
  %% AR4
  role(manager) : cap(banned(Agent)) => enact(manager).

  % COORDINATION
  %% AR5
  send(_, tell, org(rea(Me, Role)))
    : bel(agent(A), A \= Me), ~bel(sent(A, org(rea(Me,Role))))
    => send(A, org(rea(Me, Role))).
  %% AR6
  send(manager, achieve, bel(verified(Me)))
    : org(rea(A, manager)), ~bel(sent(A, opt(obj(bel(verified(Me))))))
    => send(A, opt(obj(bel(verified(Me))))).
  %% AR7
  send(customer, tell, bel(verified(You)))
    : org(rea(You, customer)), ~bel(sent(You, bel(verified(You))))
    => send(You, bel(verified(You))).

  % COMMITMENT
  %% AR8
  obj(bel(verified(Agent)))
    : org(rea(Me,manager)),
      ~org(norm(Me, manager, forbidden, bel(verified(Agent)), false))
    => commit(verified(Agent)).
  %% AR9a
  obj(bel(registered(Me)))
    : goal(bought(Item))
    => commit(registered(Me)).
  %% AR9b
  obj(bel(registered(Me)))
    : goal(sold(Item))
    => commit(registered(Me)).
  %% AR10
  viol(Me, customer, obliged, bel(paidFor(Item)))
    : bel(want(Item))
    => commit(paidFor(Item)).
  %% AR11
```

```
45    viol(Ag, buyer, obliged, bel(verified(Ag)))
46      : org(rea(Me, manager)), bel(participates(Ag,_))
47      => commit(banned(Ag)).
48    %% AR12
49    true
50      : goal(bought(Item)), bel(auction(_,Item)),
51        bel(registered(Me)), bel(badInfo(Me)),
52        bel(participates(Me,Item))
53      => commit(bid(Me, Item)).
54  }
```

A.3 *Jason*-agents

The *Jason*-agents are provided as is. They are not intended to be complete *Jason*-agents able to use sophisticated strategies for participating in auctions, but rather to show how AORTA can be used to guide their behavior in an organization and provide ways to handle e.g. obligations and the potential violation thereof.

Jason project

```
1  MAS auctionhouse {
2
3    infrastructure: AORTA(organization("ah.mm"))
4    environment: aorta.auctionhouse.AuctionHouse
5
6    agents:
7      bob buyer.asl [aorta="ah.aorta"];
8      carol buyer.asl [aorta="ah.aorta"];
9      sally seller.asl [aorta="ah.aorta"];
10     mike manager.asl [aorta="ah.aorta"];
11
12 }
```

Buyer agent

```
1  want("PH-lamp").
2
3  +want(Item) <- !bought(Item).
4
5  +!registered(Agent) <- register("Address", "Account").
6
7  +!bought(Item)
8    : not(auction(_,Item,_,_,_))
9    <-  .wait({+auction(Id,Item,_,_,_)});
10     .print("Auction for ", Item, " created!");
11     enter_auction(Id);
12     !bought(Item).
13 +!bought(Item)
14   : auction(_,Item,_,_,_) & .my_name(Me) & not(verified(Me))
```

```
  <-  .wait({+verified(Me)});
    !bought(Item).
+!bought(Item)
  : auction(_,Item,_,_,_) & .my_name(Me) & verified(Me)
  <-  !bid(_, Item).

+!bid(_, Item)
  : auction(Id,Item,_,_,_) & bid_high(Id,Me,P) & .my_name(Me)
  <- .wait(1000); !bid(_, Item).
+!bid(_, Item)
  : auction(Id,Item,_,SP,_) & not(bid_high(Id,_,_))
  <- .print("Bidding ", SP, " on ", Item); bid(Id, SP); !bid(_,Item).
+!bid(_, Item)
  : auction(Id,Item,_,_,_) & bid_high(Id,_,P)
  <- .print("Bidding ", P+5, " on ", Item); bid(Id, P + 5); !bid(_,Item)
      .

-participant(Id, Me)
  : .my_name(Me) & auction(Id,Item,_,_,_) & .intend(bid(_,Item))
  <- .drop_intention(bid(_,Item)); .print("Giving up on ", Item).

+won(Id) : auction(Id,Item,_,_,_)
  <- .drop_intention(bid(_,Item)); .print("Won ", Item);
    +bought(Item); leave_auction(Id).

+!paidFor(Item)
  : auction(Id,Item,_,_,_) & won(Id)
  <- pay(Id).
```

Seller agent

```
have("PH-lamp").

+have(Item) <- !sold(Item).

+!registered(_) <- register("Address", "Account").

+!sold(Item)
  : .my_name(Me) & verified(Me)
  <- !auction(Me,Item).
+!sold(Item) <- .wait(1000); !sold(Item).

+!auction(Me,Item)
  : .my_name(Me) &
    registered(Me,_,_) &
    have(Item) &
    not(auction(_,Item,Me,_,_)) &
    Price = 10
  <-  .print("Selling ", Item, " for ", Price);
    start_auction(Item, Price, 5).
+!auction(Me,Item) <- .wait(1000); !auction(Me,Item).

// info
+auction_done(Id,no_winner,_)
```

```
  : my_auction(Id) & auction(Id,Item,_,_,_)
  <- .print(Item, " was not sold").
+auction_done(Id,_,_)
  : auction(Id,Item,Seller,_,_) & .my_name(Seller)
  <- .print("My auction for ", Item, " was successful!").

+paid(Id)
  : auction(Id,Product,Seller,_,_) & .my_name(Seller)
    & auction_done(Id,Winner,_)
  <- .print(Winner, " paid for ", Product).
+bid(Id,Bidder,Bid)
  : my_auction(Id) & auction(Id,Item,_,_,_)
  <- .print("Bid on ", Item, ": ", Bid, " by ", Bidder).
```

Manager agent

```
+!verified(Agent)
  : not(badInfo(Agent)) & registered(Agent,_,_)
  <- .print("Verifying ", Agent); verify(Agent).

+!banned(Agent)
  : participant(Id,Agent) & auction(Id,Item,_,_,_)
  <- remove_from_auction(Id,Agent);
    .print("Removing ", Agent, " from auction for ", Item).
```

A.4 2APL-agents

As with the *Jason*-agents, the 2APL agents are simple, proof of concept agents that exhibit basic functionality.

2APL project

```
<apaplmas>
  <environment name="ah" file="auctionhouse-2apl.jar"/>
  <aorta organization="ah.mm" />
  <agent name="bob" file="buyer.2apl" aorta="ah.aorta" />
  <agent name="carol" file="buyer.2apl" aorta="ah.aorta" />
  <agent name="sally" file="seller.2apl" aorta="ah.aorta" />
  <agent name="mike" file="manager.2apl" aorta="ah.aorta" />
</apaplmas>
```

Common module

This module contains ingredients necessary for all the agents and each include it as part of their program.

```
beliefupdates:
  { bid_high(OldId, OldAg, OldBid) } NewBid(Id,Ag,Bid)
    { not bid_high(OldId, OldAg, OldBid), bid_high(Id,Ag,Bid) }
  { true } NewBid(Id,Ag,Bid) { bid_high(Id,Ag,Bid) }

```

```
pcrules:
  event(badInfo(Ag), ah) <- true | { +badInfo(Ag); }
  event(registered(Ag,X,Y), ah) <- true |
    { +registered(Ag); +registered(Ag,X,Y); }
  event(auction(Id,Item,Owner,StartPrice,EndTime), ah) <- true |
    { +auction(Id,Item,Owner,StartPrice,EndTime); }
  event(bid_high(Id,Bidder,Price), ah) <- true |
    { NewBid(Id,Bidder,Price); }
  event(bid(Id,Bidder,Price), ah) <- true | { +bid(Id,Bidder,Price); }
  event(auction_done(Id,Winner,Bid), ah) <- true |
    { +auction_done(Id,Winner,Bid); }
  event(participant(Id,Agent), ah) <- true | { +participant(Id,Agent); }
```

Buyer agent

```
include: common.2apl;

beliefs:
  want(lamp).

pgrules:
  <- want(X) | { adopta(bought(X)); }
  registered(Me) <- true | { @ah(register(address,account), S); }

  bought(Item) <- auction(Id,Item,_,_,_) | { @ah(enterAuction(Id), S); }
  bought(Item) <- auction(Id,Item,_,_,_) and me(Me) and verified(Me) |
    { adopta(bid(Me,Item)); }

  bid(bob, Item) <- auction(Id,Item,_,_,_) | { @ah(bid(Id,10), S); }
  bid(carol, Item) <- auction(Id,Item,_,_,_) and bid_high(Id,bob,_) |
    { @ah(bid(Id,15), S); }
  bid(carol, Item) <- auction(Id,Item,_,_,_) | { @ah(bid(Id,5), S); }

  paidFor(Item) <- auction(Id,Item,_,_,_) and won(Id) |
    { @ah(pay(Id), S); }

pcrules:
  event(neg(participant(Id,Me)), ah)
    <- me(Me) and auction(Id,Item,_,_,_) |
    { if G(bid(Me,Item)) dropgoal(bid(Me,Item)); }

  event(won(Id), ah) <- auction(Id,Item,_,_,_) |
    { dropgoal(bid(Me,Item)); +bought(Item); @ah(leaveAuction(Id), S); }
```

Seller agent

```
include: common.2apl;

beliefs:
  have(lamp).

```

```
pgrules:
  <- have(X) | { adopta(sold(X)); }
  registered(Me) <- true | { @ah(register(address,account), S); }
  sold(Item) <- me(Me) and verified(Me) | { adopta(auction(Me,Item)); }
  auction(_, Item)
    <- me(Me) and registered(Me,_,_) and have(Item) and
      not auction(_,Item,Me,_,_) |
    { @ah(startAuction(Item, 5, 10), S); }
```

Manager agent

```
include: common.2apl;

pgrules:
  verified(Agent) <- registered(Agent) and not badInfo(Agent) |
    { @ah(verify(Agent), S); B(S = ok); +verified(Agent); }

  banned(Agent) <- participant(Id,Agent) and auction(Id,_,_,_,_) |
    { @ah(removeFromAuction(Id,Agent), S); }
```

A.5 AIL agents

We use the Gwendolen language included with AIL to create simple, proof of concept agents that exhibit basic functionality.

AIL project

ah.ail

```
env = auctionhouse.AuctionHouse

mas.file = ah.aortasetup
mas.builder = aorta.AortaMASBuilder
```

ah.aortasetup

```
mas.file = agents.gwen
mas.builder = gwendolen.GwendolenMASBuilder

aorta.model = ah.mm
aorta.rules =  ah.aorta
```

Buyer agent

```
:Initial Beliefs:
badInfo(bob)
want(lamp)

:Initial Goals:
```

```
:Plans:
+!registered(Me) [achieve] : { ~B registered(Me) }
  <- register(address, account), +!registered(Me) [achieve];

+!bought(Item) [achieve] : { B auction(Id,Item,C,SP,ET), B me(Me) }
  <- *verified(Me), +!bid_done(Me, Item) [achieve],
    -!bought(Item) [achieve];

+!bid_done(Me,Item) [achieve] : { B won(Ag,Item) }
  <- -!bid_done(Me,Item) [achieve];
+!bid_done(Me,Item) [achieve]
  : { B auction(Id,Item,Ag,SP,ET), B bid_high(Id,Me,Bid) }
  <- *bid_high(Id,carol,BobBid), +!bid_done(Me,Item) [achieve];
+!bid_done(Me,Item) [achieve] : { B auction(Id,Item,Ag,SP,ET) }
  <- bid(Id), +!bid_done(Me,Item) [achieve];

-participates(Me,Item) : { B me(Me) }
  <- -!bought(Item) [achieve], -!bid_done(Me,Item) [achieve];

+won(Id) : { B auction(Id,Item,A,SP,ET) }
  <- -!bid_done(Me,Item) [achieve], +bought(Item), leave_auction(Id);

+!paidFor(Item) [achieve]
  : { B auction(Id,Item,A,SP,ET), B me(Me), B won(Me,Item) }
  <- pay(Id), -!paidFor(Item) [achieve];

+auction(Id,Item,Ag,SP,ET) : { B want(Item) }
  <- +!bought(Item) [achieve];
+registered(Me)
  : { B me(Me), B auction(Id,Item,Ag,SP,ET), G bought(Item) [achieve] }
  <- enter_auction(Id);
```

Seller agent

```
:Initial Beliefs:
badInfo(bob)
have(lamp)

:Initial Goals:
sold(lamp) [achieve]

:Plans:
+!registered(Me) [achieve] : { ~B registered(Me) }
  <- register(address, account), +!registered(Me) [achieve];

+!sold(Item) [achieve] : { B me(Me) }
  <- *verified(Me), +!auction_created(Me,Item) [achieve];

+!auction_created(Me,Item) [achieve] : { ~B auction(Me,Item) }
  <- start_auction(Item, 10, 1), *auction(Id,Item,Me,SP,ET),
    *auction_done(Id,Winner,Bid), +sold(Item);
```

Manager agent

```
:Initial Beliefs:
badInfo(bob)

:Initial Goals:

:Plans:
+!verified(Agent) [achieve] : { ~B badInfo(Agent), B registered(Agent) }
  <- verify(Agent);
+!verified(Agent) [achieve] : { B badInfo(Agent) }
  <- -!verified(Agent) [achieve];
+!banned(Agent) [achieve]
  : { B participates(Agent,Item), B auction(Id,Item,A,SP,ET) }
    <- remove_from_auction(Id,Agent), +banned(Agent),
    -!banned(Agent) [achieve];
```

APPENDIX B

Encapsulating Organizational Models

This appendix aims to show that the AORTA metamodel can capture the concepts of different organizational models, thus making it possible for agents to use the AORTA metamodel to reason about an organization specified in a different language.

As we will see, some concepts of the organizational models are not easily translated to the metamodel. However, we argue that simplicity is the key, and that we in this way enable the use of AORTA with different organizational models. In some cases, this is more beneficial than having to implement different reasoning strategies for different models, since the AORTA reasoning programs will be usable for each translated model.

The first part of the appendix is dedicated to our translation of the OperA model (Appendix B.1) and the second part describes how to translate the MOISE$^+$ model (Appendix B.2).

B.1 The OperA Model

In this section, we show that a subset of the OperA organizational model can be translated into the AORTA metamodel. As described in Chapter 2, the OperA model is a highly expressive model with a logical foundation, designed such that agents' desires and goals are distinguishable from the organizational aims. It consists of three parts: an organizational model (OM), a social model (SM), and an interaction model (IM). We focus on the OM, which describes the organizational structure and objectives in terms of groups, roles, norms and scenes. The other models are more concerned with operationalization and can as such not be directly compared to the AORTA metamodel.

In some ways, the OperA model is richer than the AORTA metamodel. The notion of, e.g., groups and scenes is not present in AORTA, and they cannot be directly translated into an equivalent notion[1]. In the following we focus on the social, interaction and normative structure, and show how a subset of the OperA model can be translated into the AORTA metamodel. We do not directly consider the communicative structure, but since the ontology defined here is used in the other structures, the communication language is inherently included in the AORTA metamodel as well.

In each subsection, we provide an example of how the translation is done using the EAH scenario. This translation is based on our implementation of the EAH scenario in OperettA [Aldewereld and Dignum, 2011] and was translated using the translation feature in the AORTA IDE (see Chapter 7).

[1] It is possible to, e.g., create multiple versions of a role with different names; that is, for a role r and groups $g1$ and $g2$, we can create two roles, r_g1 and r_g2.

The rest of this section explains how to translate the social structure, the interaction structure and the normative structure into the AORTA metamodel.

B.1.1 Social structure

The social structure specifies the roles, groups and role dependencies of the organization. Since the metamodel does not contain the notion of groups, we show only how to translate roles and role dependencies. A role in OperA is a tuple

$$role(r, Obj, Sbj, Rgt, Nor, tp),$$

where r is the role identifier, *Obj* and *Sbj* are objectives and sub-objectives, *Rgt* is the set of rights associated with the role, *Nor* is the set of norms and *tp* is the type of the role (which can be either external or institutional).

The metamodel has no concept of rights and does not distinguish between different role types. A role can therefore at most be described by its identifer, objectives, sub-objectives and norms. An OperA role then becomes role(r, *Obj*) in the metamodel. We show how role norms can be translated in the normative structure. A set of sub-objectives for objective γ is defined in OperA as a set $\Pi\gamma = \{\gamma_1, \ldots, \gamma_n\}$ such that $\bigwedge_{i=1}^{n} \gamma_i \rightarrow \gamma$. For each objective, γ, we thus add obj($\gamma, \Pi\gamma$) to the metamodel.

The social structure defines three kinds of dependency relations: hierarchical, market and network relations. They differ in how agents in the relation have authority over one another. In AORTA, the dependency relation is hierarchical, letting agents enacting one role delegate tasks to other agents. We thus perform the following translation: an OperA hierarchical relation $r_1 \succeq_{\gamma}^{H} r_2$ becomes dep(r_1, r_2, γ) in the metamodel.

Example B.1 (Translating the social structure)**.** *Consider the social structure of the EAH as we defined it in Chapter 2 (Figure 2.1). From this, we can extract AORTA roles, objectives and dependencies using the rules described above.*

The system consists of four roles with a number of objectives. This translates into the following predicates:

role(*customer*, {*boughtItem*(*Item*), *soldItem*(*Item*),
registered(*Agent*), *paid*(*Id*), *delivered*(*Id*)})
role(*buyer*, {*bid*(*Id*, *Agent*), *verified*(*Agent*)})
role(*seller*, {*auction*(*Id*, *Item*, *Agent*), *verified*(*Agent*)})
role(*manager*, {*verified*(*Agent*)})

All of the objectives in the scenario are not present in the figure, as is evident from the objectives of each role. For example, the customer's objectives to buy or sell an item. The objectives are represented by the following predicates:

obj(*boughtItem*(*Item*), {*paid*(*Id*), *delivered*(*Id*), *bid*(*Id*, *Agent*)})
obj(*soldItem*(*Item*), {*paid*(*Id*), *delivered*(*Id*), *auction*(*Id*, *Item*, *Agent*)})

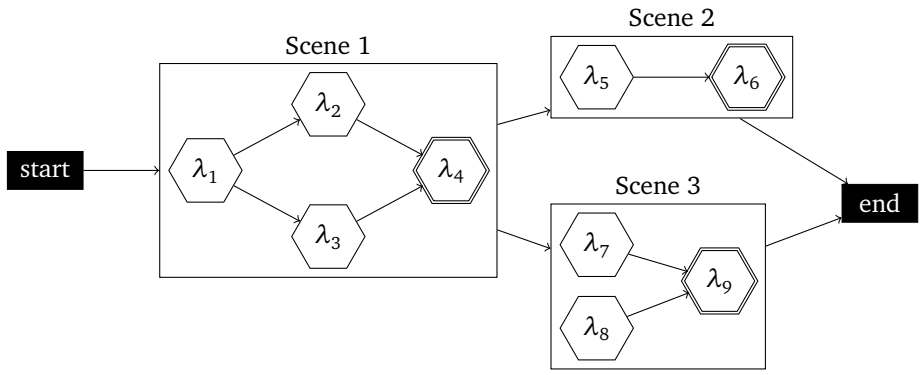


Figure B.1: Interaction structure with landmark patterns for each scene script. Landmarks with a double edge are results of the scene script.

$$\mathsf{obj}(\mathit{registered}(\mathit{Agent}), \{\})$$
$$\mathsf{obj}(\mathit{paid}(\mathit{Id}), \{\})$$
$$\mathsf{obj}(\mathit{delivered}(\mathit{Id}), \{\})$$
$$\mathsf{obj}(\mathit{bid}(\mathit{Id}, \mathit{Agent}), \{\mathit{verified}(\mathit{Agent})\})$$
$$\mathsf{obj}(\mathit{auction}(\mathit{Id}, \mathit{Item}, \mathit{Agent}), \{\mathit{verified}(\mathit{Agent})\})$$
$$\mathsf{obj}(\mathit{verified}(\mathit{Agent}), \{\mathit{registered}(\mathit{Agent})\})$$

Finally, the agents depend on each other for the completion of certain objectives. This translates into the following predicates:

$$\mathsf{dep}(\mathit{customer}, \mathit{manager}, \mathit{verified}(\mathit{Agent}))$$
$$\mathsf{dep}(\mathit{seller}, \mathit{buyer}, \mathit{bid}(\mathit{Id}, \mathit{Agent}))$$
$$\mathsf{dep}(\mathit{buyer}, \mathit{seller}, \mathit{auction}(\mathit{Id}, \mathit{Item}, \mathit{Agent}))$$
$$\mathsf{dep}(\mathit{customer}, \mathit{seller}, \mathit{delivered}(\mathit{Id}))$$
$$\mathsf{dep}(\mathit{customer}, \mathit{buyer}, \mathit{paid}(\mathit{Id}))$$

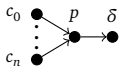
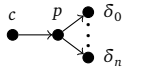
B.1.2 Interaction structure

The interaction structure divides the organizational activity into scene scripts that provide partial ordering of objectives and transitions between scenes that provide synchronization and evolution of roles. Objectives of scenes are partially ordered using landmark patterns, which represent the minimum requirements for achieving the results of the scene.

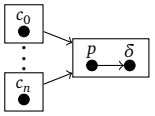
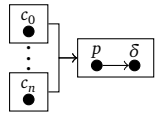
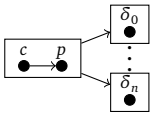
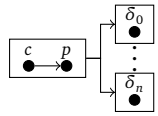
Figure B.1 shows an interaction structure with three scene scripts. We use this to show how to perform the translation into the metamodel. Informally, the interaction structure tells us that scene 1 ought to be completed before beginning on scene 2 and 3, and either scene 2 or scene 3 ought to be completed before moving on to the end scene. Furthermore,

Table B.2: Different types of landmark patterns and scene transitions and their corresponding conditional obligations with deadline.

Landmark patterns

	$O(p < \delta \mid c_0 \wedge \cdots \wedge c_n)$		$O(p < \delta_0 \vee \cdots \vee \delta_n \mid c)$

Scene transitions

	$O(p < \delta \mid c_0 \vee \cdots \vee c_n)$		$O(p < \delta \mid c_0 \wedge \cdots \wedge c_n)$
	$O(p < \delta_0 \vee \cdots \vee \lambda_n \mid c)$		$O(p < \delta_0 \vee \cdots \vee \lambda_n \mid c)$

looking at scene 1, landmark λ_1 should be completed before λ_2 and λ_3, and so on. Since AORTA does not have a notion of scenes, we propose a translation of the interaction structure to conditional obligations with deadline. We basically convert each landmark pattern to a set of conditional obligations, and furthermore connect the scenes using conditional obligations as well.

An obligation $O(p < \delta \mid c)$ means that p ought to be achieved once c is achieved, and before δ occurs. We thus propose a translation where each landmark is the objective of an obligation, with the previous landmark(s) as condition and the next landmark(s) as deadline. For example, given a landmark pattern $\lambda_a \leq \lambda_b \leq \lambda_c$, the obligation to achieve λ_b becomes $O(\lambda_b < \lambda_c \mid \lambda_a)$, which states that the agent ought to achieve λ_b once λ_a has been achieved, but before λ_c is achieved, corresponding to the landmark pattern. In other patterns where multiple landmarks must be achieved in parallel before the next landmark, the obligation becomes a bit more complex, since we have to incorporate this. However, the well-defined semantics of landmark patterns [Kumar et al., 2002] and of scene transitions makes it possible to translate such patterns into conditional obligations that correspond well with the meaning of the pattern. Table B.2 lists the different kinds of patterns that can appear in the interaction structure.

We deal with the edge cases as follows. If for a landmark λ there is no landmark λ', such that $\lambda' \leq \lambda$, then the condition for the obligation to achieve λ is $\top$. This corresponds to an obligation that is immediately activated. If for a landmark λ there is no λ', such that $\lambda \leq \lambda'$, then the deadline for the obligation to achieve λ is $\bot$. This corresponds to an obligation without a deadline.

Example B.2. *We can use the translation scheme above to translate the interaction structure in Figure B.1. The first objective, λ_1, is translated into the obligation $O(\lambda_1 < \lambda_2 \vee \lambda_3 \mid \top)$. The condition is $\top$ since λ_1 is the first objective. The deadline is reached when either λ_2 or*

λ_3 have been achieved, corresponding the fact that they are after λ_1 in the partial ordering. The rest of the resulting obligations are shown below.

$$O(\lambda_2 < \lambda_4 \mid \lambda_1) \qquad O(\lambda_3 < \lambda_4 \mid \lambda_1)$$
$$O(\lambda_4 < \lambda_5 \vee \lambda_7 \vee \lambda_8 \mid \lambda_2 \wedge \lambda_3) \qquad O(\lambda_5 < \lambda_6 \mid \lambda_4)$$
$$O(\lambda_6 < \bot \mid \lambda_5) \qquad O(\lambda_7 < \lambda_9 \mid \lambda_4)$$
$$O(\lambda_8 < \lambda_9 \mid \lambda_4) \qquad O(\lambda_9 < \bot \mid \lambda_7 \wedge \lambda_8)$$

The translation shown above omits a few parts of the interaction structure. Transitions between scenes may include *role evolution* relations, specifying how agents can (or are obliged) to enact a role in the next scene based on there current role. Furthermore, a role evolution may specify a *conflict* between roles, i.e., that two roles may not be simultaneously enacted by a single agent. This is not captured in the translation above. A role evolution relation is defined by

$$\textit{role-evolution}(s.r_1, t.r_2, SN, \lambda),$$

where s and t are scenes, r_1 is a role in s, r_2 a role in t, *SN* is the type of relation and can be either *necessary*, *sufficient* or *conflict*, and λ is the set of conditions for performing the role evolution (the landmarks that must be fulfilled by the agent). As we are considering only obligations, we are not translating sufficient role evolutions (i.e., the agent is allowed to enact a certain role in the next scene).

A necessary role evolution is translated into the following conditional obligation:

$$O(\mathsf{rea}(Ag, r_2) < \lambda_t \mid \mathsf{rea}(Ag, r_1) \wedge \lambda),$$

where Ag is is the agent, λ_t is the set of initial landmarks of scene t, and λ is the set of conditions for performing the role evolution. A role conflict is translated into the following conditional prohibition:

$$F(\mathsf{rea}(Ag, r_2) < \lambda_t^r \mid \mathsf{rea}(Ag, r_1) \wedge \lambda),$$

where Ag is the agent, λ_t^r is the set of results of scene t, and λ is the set of conditions for the role evolution. A global role conflict can be translated into $F(\mathsf{rea}(Ag, r_2) < \bot \mid \mathsf{rea}(Ag, r_1))$.

Example B.3 (Translating the interaction structure)**.** *Consider the interaction structure of the EAH as we defined it in Chapter 2 (Figure 2.2). From this, we can create a number of AORTA conditional obligations that represent the transition between landmarks in scenes and between different scenes (i.e., between the last landmark in a scene and the first in the next). For example, the registration scene can be represented by the following set of conditional norms:*

$$\mathsf{cond}(\textit{customer}, \textit{obliged}, \textit{registered}(\textit{Agent}), \textit{verified}(\textit{Agent}), \top)$$
$$\mathsf{cond}(\textit{buyer}, \textit{obliged}, \textit{verified}(\textit{Agent}), \textit{bid}(\textit{Id}, \textit{Winner}) \vee \textit{auction}(\textit{Id}, \textit{Item}, \textit{Agent}),$$

$$
\begin{aligned}
&\qquad\qquad me(Agent) \wedge registered(Agent))\\
&\text{cond}(seller, obliged, verified(Agent),\\
&\qquad\qquad bid(Id, Winner) \vee auction(Id, Item, Agent),\\
&\qquad\qquad me(Agent) \wedge registered(Agent))\\
&\text{cond}(manager, obliged, verified(Agent),\\
&\qquad\qquad bid(Id, Winner) \vee auction(Id, Item, Agent),\\
&\qquad\qquad me(Agent) \wedge registered(Agent))
\end{aligned}
$$

As we can see, the conditional norms are bound to the agents responsible for the objectives in the norm. For example, the customer is obliged to complete the registration-objective, whereas both buyers, sellers and managers are obliged to complete the verification-objective.

B.1.3 Normative structure

The normative structure defines the expected boundaries of the agents participating in an organization. This is done by specifying *norms* that describe what agents enacting specific roles are expected to do. Norms in the normative structure are divided into *role norms*, *scene norms* and *transition norms*. Role norms define the generally expected behavior of a role regardless of participation in scenes. Scene norms define the expected behavior of roles participating in a specific scene. Transition norms define the limitations related with enacting new roles when moving between scenes, and were handled above in the translation of the interaction structure.

Norms in OperA are specified in Logic for Contract Representation (LCR). Different types of obligations are defined: conditional obligations, obligations with deadline and obligations without deadline. A obligation to achieve p before δ when c in LCR is translated to $O(p < \delta \mid c)$ in the metamodel. If $c = \top$, the obligation has no condition. If $\delta = \bot$, the obligation has no deadline. We can make a similar translation of OperA prohibitions. While the semantics of norms in LCR differs slightly from the AORTA norms, the intended meaning is the same. Therefore, we believe that the straightforward translation is reasonable.

Example B.4 (Translating the normative structure)**.** *Since norms are straightforwardly translated from LCR to AORTA conditional norms, we simply list a few of the norms of the EAH that were defined in the OperA model.*

$$
\begin{aligned}
&\text{cond}(manager, obliged, \neg auction(Id, Item, Agent), auction\_done(Id),\\
&\qquad\qquad viol(Agent, seller, verified(Agent)) \wedge auction(Id, Item, Agent))\\
&\text{cond}(manager, obliged, participant(Id, Agent), \top,\\
&\qquad\qquad viol(Agent, customer, paid(Id)))
\end{aligned}
$$

These norms both concern the responsibility of the manager. For example, a manager is obliged to shut down auctions that where created by unverified agents.

B.2 The Moise$^+$ Model

The MOISE$^+$ model is based on three organizational *dimensions*: the structural, functional and deontic dimensions. The structural dimension defines roles, links between roles, and groups. The functional dimension describes how the system should achieve its organizational goals by decomposing them into plans and distributing them to the agents as missions. Finally, the deontic dimension describes for each role, the missions they are permitted and obligated to commit to.

As with the OperA model, the MOISE$^+$ model may seem richer than our metamodel. The notion of groups is not present in AORTA, so our translation of the MOISE$^+$ model will not be able to capture this part of the model. On the other hand, the notion of obligations and permissions is quite simple in MOISE$^+$. In any case, care must be taken when translating the model.

In the following subsections, we show how to translate (parts of) the structural, functional and deontic dimensions of MOISE$^+$.

B.2.1 Structural specification

The structural specification contains the roles and groups of the organization along with several constraints on the roles. It is possible to define abstract roles that other roles can inherit properties from (e.g., a manager role may inherit properties from an employee role). Groups can define cardinality constraints for roles, putting a limit on the number of agents playing a certain role in the group. Finally, it is possible to define different kinds of links between agents: *acquaintance* (knows about the other agent), *communication* (is allowed to communicate) and *authority* (has authority over the other agent).

To translate the structural specification to the metamodel, we proceed as follows:

- We only consider concrete roles. Properties from abstract roles are explicitly added to the concrete role in the metamodel.
- As there is no support for groups in the metamodel, groups are not added to the metamodel.
- We consider only *authority* links in the translation: for our purpose, we assume all agents are acquainted and we pose no restrictions on the communication between the agents.
- Objectives are not present in the structural specification, but are for each role extracted from the functional and deontic specifications.
- Since cardinality constraints may conflict between groups, we generally do not include them, but if only one group is present, it is possible to specify norms to capture the constraints.

Example B.5 (Translation of the structural specification)**.** *Consider the structural specification of the EAH as shown in Figure B.3. We assume that we have extracted the objectives from the functional and deontic specification. In* MOISE$^+$*, all roles extends the* soc*-role. All of the roles are members of the* eah-group*. The labels indicate the cardinality constraints. The arrow from the manager to the customer indicates that the manager has authority over the customer.*

We thus have the following roles:

$$\mathsf{role}(\mathit{seller}, \{\mathit{soldItem}(\mathit{Item}), \mathit{auction}(\mathit{Agent}, \mathit{Item})\})$$
$$\mathsf{role}(\mathit{manager}, \{\mathit{banned}(\mathit{Agent}), \mathit{verified}(\mathit{Agent})\})$$
$$\mathsf{role}(\mathit{buyer}, \{\mathit{boughtItem}(\mathit{Item}), \mathit{bid}(\mathit{Agent}, \mathit{Item})\})$$
$$\mathsf{role}(\mathit{customer}, \{\mathit{verified}(\mathit{Agent}), \mathit{registered}(\mathit{Agent})\})$$

We use the authority link to create dependency relations. We say that for each authority link, the target role depends *on the source role for the completion of its objectives:*

$$\mathsf{dep}(\mathit{customer}, \mathit{manager}, \mathit{verified}(\mathit{Agent}))$$
$$\mathsf{dep}(\mathit{customer}, \mathit{manager}, \mathit{registered}(\mathit{Agent}))$$

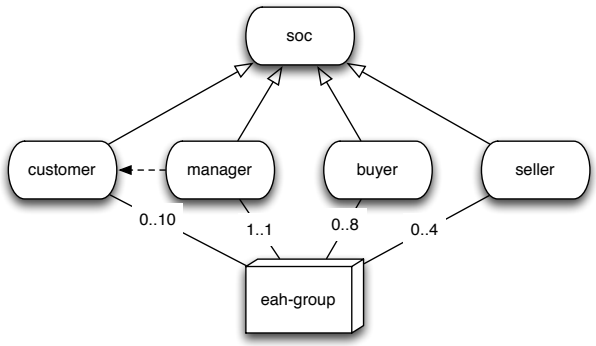


Figure B.3: *The structural specification of the EAH.*

B.2.2 Functional specification

The functional specification contains the *goals* and *missions* of the organization. Goals are described in a scheme (i.e. a tree), which decomposes the goals into subgoals by use of *plan operators*. For example, to fulfill the goal of participating in an auction, one has to fulfill a *sequence* of goals, namely registering and becoming verified. Other operators let the agents make a *choice* between subgoals and fulfill a number of subgoals in *parallel*. Furthermore, each goal may have associated a *deadline*, specifying how much time the agents are allowed to use to fulfill the goal.

The goals are related to roles in the organization via missions. Each mission defines a number of goals that contribute to the fulfillment of that mission. By committing to the mission, the agent commits to the goals of the mission.

Our translation of the functional specification works as follows:

- Each goal is an objective, with sub-objectives extracted from the schemes.
- Roles are responsible for a goal, if that goal is in a mission they role should commit to (see the deontic specification).
- To capture the goal decomposition tree, we create conditional norms for each goal:
  - The activation condition depends on the plan operator. In a sequence, the activation condition is the previous goal in the sequence. In other cases, the condition is the sub-goals (or $\top$ if the goal has no sub-goals).
  - The deadline is the conjunction of the goal's deadline and the parent goal. If none is present, the deadline is set to $\bot$.

Example B.6 (Translation of the functional specification)**.** *We consider the part of the functional specification concerned with participation and selling items (Figure B.4). The figure shows the decomposition of goals (e.g. selling an item requires to first become verified, then complete the auction for it and finally having someone pay for it), and the missions that agents may commit to (e.g., the mission to sell an item consists of the goals to sell the item and to create an auction).*

We extract the objectives and sub-objectives from the schemes:

$\mathsf{obj}(soldItem(Item), \{verified(Agent), auction\_done(Agent, Item), paidFor(Item)\})$

$\mathsf{obj}(verified(Agent), \{registered(Agent)\})$

$\mathsf{obj}(registered(Agent), \{\})$

$\mathsf{obj}(auction\_done(Item), \{auction(Agent, Item), bid(Agent, Item)\})$

$\mathsf{obj}(auction(Agent, Item), \{\})$

$\mathsf{obj}(bid(Agent, Item), \{\})$

$\mathsf{obj}(paidFor(Item), \{\})$

The following conditional norms can be extracted from the specification (based on the norms in the deontic specification that relate missions to roles):

$\mathsf{cond}(customer, obliged, registered(Agent), verified(Agent), \top)$

$\mathsf{cond}(customer, obliged, verified(Agent), auction\_done(Item), registered(Agent))$

$\mathsf{cond}(manager, obliged, verified(Agent), auction\_done(Item), registered(Agent))$

$\mathsf{cond}(seller, obliged, auction(Agent, Item), auction\_done(Agent, Item), verified(Agent))$

$\mathsf{cond}(seller, obliged, soldItem(Item), \bot, paidFor(Item))$

For example, the customer is obliged to become verified when registered before having completed their auction. Some obligations may seem strange: for example, the obligation to create

an auction when verified (why is the agent suddenly obliged to sell something?). However, the agent has decided to enact the seller role, which is responsible for selling items in the auction, so the organization expects the agent to sell an item.

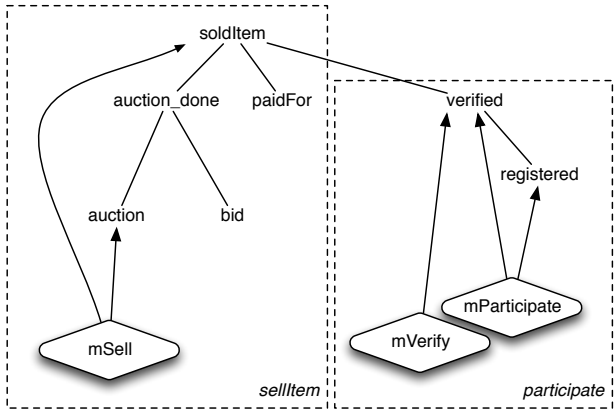


Figure B.4: *The functional specification of the EAH.*

B.2.3 Deontic specification

The deontic specification relates roles to missions by specifying the missions a role is *allowed* to commit to (permission) and is *required* to commit to (obligation). Each norm is associated with a deadline, stating how much time the agents have to complete the mission.

We translate the deontic specification as follows:

- Permissions are not part of the AORTA metamodel and are not added. However, each permission relates a role to a set of goals, and is used as such in the translation.
- Obligations are translated to conditional obligations:
  - The obligation to fulfill a mission in MOISE$^+$ translates to a conditional obligation with the conjunction of the goals in the mission.
  - The deadline is taken directly from the obligation.
  - There is no activation condition, so the norm is activated immediately when the agent enacts the role. This conforms well with the fact that the role is obliged to commit to the mission.

Example B.7 (Translation of the deontic specification)**.** *Figure B.5 shows how missions are related to roles via the deontic specification. Each mission is in this example related via a permission. The translation thus does not add any additional norms from the deontic specification.*

Had the participation mission been an obligation, the following conditional norm would be created:

$$\mathsf{cond}(\mathit{customer}, \mathit{obliged}, \mathit{verified}(\mathit{Agent}) \wedge \mathit{registered}(\mathit{Agent}), \bot, \top)$$

That is, agents enacting the customer role would be obliged to register and become verified. While this norm provides no deadline, it allows the agents to reason about the expectations toward them. By further reasoning about the norms from the functional specification, they can decide when to fulfill each objective.

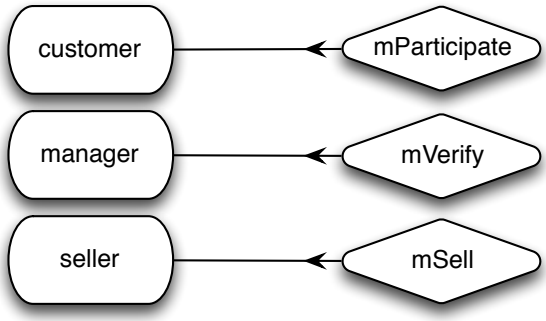


Figure B.5: *The deontic specification of the EAH.*

Bibliography

3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 19-23 August 2004, New York, NY, USA (2004). IEEE Computer Society.

Ahlbrecht, T., Bender-Saebelkampf, C., Brito, M. de, Christensen, N. C., Dix, J., Franco, M. R., Heller, H., Hess, A. V., HeSSler, A., Hübner, J. F., Jensen, A. S., Johnsen, J. B., Köster, M., Li, C., Liu, L., Morato, M. M., Ørum, P. B., Schlesinger, F., Schmitz, T., Sichman, J., Souza, K. S. de, Uez, D. M., Villadsen, J., Werner, S., Woller, Ø. G., and Zatelli, M. R. (2013). "Multi-agent programming contest 2013: the teams and the design of their systems". English. In: *Engineering Multi-Agent Systems*. Ed. by Cossentino, M. et al. Vol. 8245. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 366–390.

Alchourrón, C. E., Gärdenfors, P., and Makinson, D. (1985). "On the logic of theory change: partial meet contraction and revision functions". In: *The Journal of Symbolic Logic* 50.2, pp. 510–530.

Aldewereld, H. (2009). "Autonomy vs. Conformity: An Institutional Perspective on Norms and Protocols". PhD thesis. Utrecht.

Aldewereld, H., Dignum, F., García-Camino, A., Noriega, P., Rodríguez-Aguilar, J., and Sierra, C. (2007). "Operationalisation of norms for electronic institutions". In: *Coordination, Organizations, Institutions, and Norms in Agent Systems II*. Ed. by Noriega, P. et al. Vol. 4386. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 163–176.

Aldewereld, H. and Dignum, V. (2011). "OperettA: organization-oriented development environment". In: *Languages, Methodologies, and Development Tools for Multi-Agent Systems*. Ed. by Dastani, M. et al. Vol. 6822. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 1–18.

Aldewereld, H., Dignum, V., Jonker, C., and Riemsdijk, M. B. van (2012). "Agreeing on role adoption in open organisations". In: *KI - Künstliche Intelligenz* 26 (1), pp. 37–45.

Alechina, N., Dastani, M., and Logan, B. (2012). "Programming Norm-Aware Agents". In: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'12)*. Ed. by Hoek, W. van der et al. IFAAMAS, pp. 1057–1064.

Andrighetto, G. et al., eds. (2013). *Normative multi-agent systems*. Dagstuhl Follow-Ups. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

Arcos, J. L., Esteva, M., Noriega, P., Rodríguez-Aguilar, J. A., and Sierra, C. (2005). "An integrated development environment for electronic institutions". In: *Software Agent-Based Applications, Platforms and Development Kits*. Ed. by Unland, R. et al. Whitestein Series in Software Agent Technologies. Birkhäuser Basel, pp. 121–142.

Astefanoaei, L., Dastani, M., Meyer, J.-J., and Boer, F. S. de (2008). "A verification framework for normative multi-agent systems". In: *Intelligent and Multi-Agent Systems*. Ed. by Bui, T. D. et al. Vol. 5357. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 54–65.

Baier, C. and Katoen, J.-P. (2008). *Principles of Model Checking*. Representation and Mind Series. The MIT Press.

Balke, T. et al., eds. (2014). *Coordination, Organizations, Institutions, and Norms in Agent Systems IX - COIN 2013 International Workshops, COIN@AAMAS, St. Paul, MN, USA, May 6, 2013, COIN@PRIMA, Dunedin, New Zealand, December 3, 2013, Revised Selected Papers*. Vol. 8386. Lecture Notes in Computer Science. Springer.

Bazzan, A. L. et al., eds. (2014). *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014*. IFAAMAS/ACM.

Behrens, T., Hindriks, K., and Dix, J. (2011). "Towards an environment interface standard for agent platforms". English. In: *Annals of Mathematics and Artificial Intelligence* 61.4, pp. 261–295.

Belnap, N., Perloff, M., and Xu, M. (2001). *Facing the Future: Agents and Choices in Our Indeterminist World*. Oxford University Press.

Boella, G. and Torre, L. van der (2004). "Regulative and Constitutive Norms in Normative Multiagent Systems." In: *KR* 4, pp. 255–265.

Boissier, O. and Riemsdijk, M. B. van (2013). "Organisational reasoning agents". In: *Agreement Technologies*. Ed. by Ossowski, S. Vol. 8. Law, Governance and Technology Series. Dordrecht: Springer Netherlands, pp. 309–320.

Bordini, R. H., Dennis, L. A., Farwer, B., and Fisher, M. (2008). "Automated Verification of Multi-Agent Programs". In: *23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), 15-19 September 2008, L'Aquila, Italy*. IEEE, pp. 69–78.

Bordini, R. H., Fisher, M., Visser, W., and Wooldridge, M. (2004a). "Verifiable multi-agent programs". In: *Programming Multi-Agent Systems*. Ed. by Dastani, M. et al. Vol. 3067. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 72–89.

Bordini, R. H., Fisher, M., Visser, W., and Wooldridge, M. (2006). "Verifying multi-agent programs by model checking". In: *Autonomous Agents and Multi-Agent Systems* 12.2, pp. 239–256.

Bordini, R. H., Fisher, M., Wooldridge, M., and Visser, W. (2004b). "Model checking rational agents". In: *IEEE Intelligent Systems* 19.5, pp. 46–52.

Bordini, R. H. and Moreira, Á. F. (2004). "Proving BDI properties of agent-oriented programming languages: the asymmetry thesis principles in AgentSpeak (L)". In: *Annals of Mathematics and Artificial Intelligence* 42.1–3, pp. 197–226.

Bordini, R. H., Wooldridge, M., and Hübner, J. F. (2007). *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley Series in Agent Technology. John Wiley & Sons.

Boutilier, C. (1994). "Toward a logic for qualitative decision theory". In: *Proceedings of the Fourth International Conference on Knowledge Representation and Reasoning (KR'94)*, pp. 75–86.

Bratman, M. E. (1987). *Intention, Plans, and Practical Reason*. Center for the Study of Language and Information.

Brazier, F., Jonker, C., and Treur, J. (2000). "Compositional design and reuse of a generic agent model". In: *Applied Artificial Intelligence* 14.5, pp. 491–538.

Broersen, J., Dastani, M., Hulstijn, J., Huang, Z., and Torre, L. van der (2001). "The boid architecture: conflicts between beliefs, obligations, intentions and desires". In: *Proceedings of the fifth international conference on Autonomous agents*. ACM, pp. 9–16.

Broersen, J., Dignum, F., Dignum, V., and Meyer, J.-J. (2004). "Designing a deontic logic of deadlines". In: *Deontic Logic in Computer Science*. Ed. by Lomuscio, A. and Nute, D. Vol. 3065. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 43–56.

Carabelea, C., Boissier, O., and Castelfranchi, C. (2005). "Using social power to enable agents to reason about being part of a group". In: *Engineering Societies in the Agents World V*. Ed. by Gleizes, M.-P. et al. Vol. 3451. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 166–177.

Castelfranchi, C., Dignum, F., Jonker, C. M., and Treur, J. (2000). "Deliberate normative agents: principles and architecture". In: *Intelligent Agents VI. Agent Theories, Architectures, and Languages*. Ed. by Jennings, N. R. and Lespérance, Y. Vol. 1757. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 364–378.

Cohen, P. R. and Levesque, H. J. (1990). "Intention is choice with commitment". In: *Artificial Intelligence* 42.2–3, pp. 213–261.

Conte, R. and Sichman, J. (1995). "DEPNET: how to benefit from social dependence". In: *Journal of Mathematical Sociology* 20, pp. 161–177.

Conte, R., Falcone, R., and Sartor, G. (1999). "Introduction: agents and norms: how to fill the gap?" In: *Artificial Intelligence and Law* 7.1, pp. 1–15.

Courcoubetis, C., Vardi, M., Wolper, P., and Yannakakis, M. (1993). "Memory-efficient algorithms for the verification of temporal properties". In: *Computer-Aided Verification*. Ed. by Kurshan, R. Springer US, pp. 129–142.

Criado, N., Argente, E., Noriega, P., and Botti, V. (2010). "Towards a normative BDI architecture for norm compliance". In: *Proceedings of The Multi-Agent Logics, Languages,*

and Organisations Federated Workshops (MALLOW 2010), Lyon, France, August 30 - September 2, 2010. Ed. by Boissier, O. et al. Vol. 627. CEUR Workshop Proceedings. CEUR-WS.org.

Dastani, M. (2008). "2APL: a practical agent programming language". In: *Autonomous Agents and Multi-Agent Systems* 16.3, pp. 214–248.

Dastani, M., Dignum, V., and Dignum, F. (2003). "Role-assignment in open agent societies". In: *The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings*. ACM, pp. 489–496.

Dastani, M., Grossi, D., Meyer, J.-J., and Tinnemeier, N. (2009). "Normative multi-agent programs and their logics". In: *Knowledge Representation for Agents and Multi-Agent Systems*. Ed. by Meyer, J.-J. and Broersen, J. Vol. 5605. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 16–31.

Dastani, M., Riemsdijk, M. B. van, Hulstijn, J., Dignum, F., and Meyer, J.-J. (2005). "Enacting and deacting roles in agent programming". In: *Agent-Oriented Software Engineering V*. Ed. by Odell, J. et al. Vol. 3382. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 189–204.

Davidsson, P. (2002). "Agent based social simulation: a computer science view". In: *Journal of Artificial Societies and Social Simulation* 5.1.

Dennis, L. A. and Farwer, B. (2008). "Gwendolen: a BDI language for verifiable agents". In: *Logic and the Simulation of Interaction and Reasoning (AISB'08 Workshop)*. Ed. by Löwe, B.

Dennis, L. A., Fisher, M., Webster, M. P., and Bordini, R. H. (2012). "Model checking agent programming languages". In: *Automated Software Engineering* 19.1, pp. 5–63.

Dennis, L. A., Tinnemeier, N., and Meyer, J.-J. (2010). "Model checking normative agent organisations". In: *Computational Logic in Multi-Agent Systems*. Ed. by Dix, J. et al. Vol. 6214. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 64–82.

Denti, E., Omicini, A., and Ricci, A. (2001). "tuProlog: a light-weight Prolog for internet applications and infrastructures". In: *Practical Aspects of Declarative Languages*. Ed. by Ramakrishnan, I. Vol. 1990. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 184–198.

Dignum, F., Morley, D., Sonenberg, E. A., and Cavedon, L. (2000). "Towards socially sophisticated BDI agents". In: *4th International Conference on Multi-Agent Systems, ICMAS 2000, Boston, MA, USA, July 10-12, 2000*. IEEE Computer Society, pp. 111–118.

Dignum, F., Dignum, V., Thangarajah, J., Padgham, L., and Winikoff, M. (2008). "Open agent systems ???" In: *Agent-Oriented Software Engineering VIII*. Ed. by Luck, M. and

Padgham, L. Vol. 4951. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 73–87.

Dignum, F., Kinny, D., and Sonenberg, L. (2002). "From desires, obligations and norms to goals". In: *Cognitive Science Quarterly* 2.3–4, pp. 407–430.

Dignum, V. (2004). "A Model for Organizational Interaction: Based on Agents, Founded in Logic". PhD thesis. Utrecht University.

Dignum, V. (2009). *Handbook of Research on Multi-agent Systems: Semantics and Dynamics of Organizational Models*. Information Science Reference.

Dignum, V. and Dignum, F. (2004). "What's in it for me? Agent deliberation on taking up social roles". In: *EUMAS 2004 - Proceedings of the Second European Workshop on Multi-Agent Systems, Barcelona, Spain, 2004*.

Dignum, V. and Dignum, F. (2012). "A logic of agent organizations". In: *Logic Journal of IGPL* 20.1, pp. 283–316.

Dignum, V., Meyer, J.-J., Dignum, F., and Weigand, H. (2003). "Formal specification of interaction in agent societies". In: *Formal Specification of Interaction in Agent Societies*. Ed. by Hinchey, M. et al. Vol. 2699. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 37–52.

Doan, T. T., Yao, Y., Alechina, N., and Logan, B. (2014). "Verifying heterogeneous multi-agent programs". In: *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014*. Ed. by Bazzan, A. L. et al. IFAAMAS/ACM, pp. 149–156.

Dybalova, D., Testerink, B., Dastani, M., and Logan, B. (2014). "A framework for programming norm-aware multi-agent systems". In: *Coordination, Organizations, Institutions, and Norms in Agent Systems IX*. Ed. by Balke, T. et al. Vol. 8386. Lecture Notes in Computer Science. Springer International Publishing, pp. 364–380.

Esteva, M., Cruz, D. de la, and Sierra, C. (2002). "Islander: an electronic institutions editor". In: *The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy, Proceedings*. ACM.

Esteva, M., Rosell, B., Rodríguez-Aguilar, J. A., and Arcos, J. L. (2004). "Ameli: an agent-based middleware for electronic institutions". In: *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 19-23 August 2004, New York, NY, USA*. IEEE Computer Society, pp. 236–243.

Ferber, J., Gutknecht, O., and Michel, F. (2003). "From agents to organizations: an organizational view of multi-agent systems". In: *Agent-Oriented Software Engineering IV*. Ed. by Giorgini, P. et al. Vol. 2935. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 214–230.

Fikes, R. E. and Nilsson, N. J. (1971). "STRIPS: a new approach to the application of theorem proving to problem solving". In: *Proceedings of the 2nd International Joint Confer-*

ence on Artificial Intelligence (IJCAI'71). Morgan Kaufmann Publishers Inc., pp. 608–620.

Frøsig, A. and Andersen, K. B. (2015). "Organizing Heterogeneous Agents". MSc thesis. Technical University of Denmark.

Gerth, R., Peled, D., Vardi, M. Y., and Wolper, P. (1995). "Simple on-the-fly automatic verification of linear temporal logic". In: *International Symposium on Protocol Specification, Testing and Verification*, pp. 3–18.

González-Alarcón, C. A., Grimaldo, F., and Guerra-Hernández, A. (2013). "Jason intentional learning: an operational semantics". In: *Progress in Artificial Intelligence*. Ed. by Correia, L. et al. Vol. 8154. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 432–443.

Grossi, D., Aldewereld, H., and Dignum, F. (2007). "Ubi lex, ibi poena: designing norm enforcement in e-institutions". In: *Coordination, Organizations, Institutions, and Norms in Agent Systems II*. Ed. by Noriega, P. et al. Vol. 4386. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 101–114.

Grossi, D., Meyer, J.-J., and Dignum, F. (2006). "Counts-as: classification or constitution? an answer using modal logic". In: *Deontic Logic and Artificial Normative Systems*. Ed. by Goble, L. and Meyer, J.-J. Vol. 4048. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 115–130.

Hindriks, K. V. (2009). "Programming rational agents in GOAL". In: *Multi-Agent Programming: Languages, Tools and Applications*. Ed. by El Fallah Seghrouchni, A. et al. Springer US, pp. 119–157.

Hindriks, K. V., Boer, F. S. de, Hoek, W. van der, and Meyer, J.-J. (1999). "Agent programming in 3APL". In: *Autonomous Agents and Multi-Agent Systems* 2.4, pp. 357–401.

Hindriks, K. V., Boer, F. S. de, Hoek, W. van der, and Meyer, J.-J. (2001). "Agent programming with declarative goals". In: *Intelligent Agents VII Agent Theories Architectures and Languages*. Ed. by Castelfranchi, C. and Lespérance, Y. Vol. 1986. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 228–243.

Hormazábal, N., Cardoso, H. L., Rosa, J. L. de la, and Oliveira, E. (2010). "An approach for virtual organisations' dissolution". In: *Coordination, Organizations, Institutions and Norms in Agent Systems V*. Ed. by Padget, J. et al. Vol. 6069. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 70–85.

Hübner, J. F., Boissier, O., Kitio, R., and Ricci, A. (2010). "Instrumenting multi-agent organisations with organisational artifacts and agents". In: *Autonomous Agents and Multi-Agent Systems* 20.3, pp. 369–400.

Hübner, J. F., Boissier, O., and Vercouter, L. (2009). "Instrumenting multi-agent organisations with reputation artifacts". In: *Coordination, Organizations, Institutions and*

Norms in Agent Systems IV. Ed. by Hübner, J. et al. Vol. 5428. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 96–110.

Hübner, J. F., Sichman, J. S., and Boissier, O. (2002). "A model for the structural, functional, and deontic specification of organizations in multiagent systems". In: *Advances in Artificial Intelligence*. Ed. by Bittencourt, G. and Ramalho, G. Vol. 2507. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 118–128.

Hübner, J. F., Sichman, J. S., and Boissier, O. (2006). "S-moise+: a middleware for developing organised multi-agent systems". In: *Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems*. Ed. by Boissier, O. et al. Vol. 3913. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 64–77.

Hübner, J. F., Sichman, J. S., and Boissier, O. (2007). "Developing organised multiagent systems using the MOISE+ model: programming issues at the system and agent levels". In: *International Journal of Agent-Oriented Software Engineering* 1.3, pp. 370–395.

Huget, M.-P., Esteva, M., Phelps, S., Sierra, C., and Wooldridge, M. (2002). "Model checking electronic institutions". In: *Proceedings of the Workshop on Model Checking and Artificial Intelligence (MoChArt'02)*, pp. 51–58.

"ISO/IEC/IEEE Systems and software engineering – Architecture description" (2011). In: *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pp. 1–46.

Jensen, A. S. (2013a). "Deciding between conflicting influences". In: *Engineering Multi-Agent Systems*. Ed. by Cossentino, M. et al. Vol. 8245. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 137–155.

Jensen, A. S. (2013b). "On programming organization-aware agents". In: *Twelfth Scandinavian Conference on Artificial Intelligence, SCAI 2013, Aalborg, Denmark, November 20-22, 2013*. Ed. by Jaeger, M. et al. Vol. 257. IOS Press, pp. 287–290.

Jensen, A. S. (2015). "Model checking AORTA: verification of organization-aware agents". In: *ArXiv e-prints* 1503.05317.

Jensen, A. S., Aldewereld, H., and Dignum, V. (2013a). "Dimensions of organizational coordination". In: *Proceedings of the 25th Benelux Conference on Artificial Intelligence*, pp. 80–87.

Jensen, A. S. and Dignum, V. (2014). "AORTA: adding organizational reasoning to agents". In: *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014*. Ed. by Bazzan, A. L. et al. IFAAMAS/ACM, pp. 1493–1494.

Jensen, A. S., Dignum, V., and Villadsen, J. (2014). "The AORTA architecture: integrating organizational reasoning in *Jason*". In: *Engineering Multi-Agent Systems*. Ed. by Dalpiaz, F. et al. Vol. 8758. Lecture Notes in Computer Science. Springer International Publishing, pp. 127–145.

Jensen, A. S., Dignum, V., and Villadsen, J. (2015a). “A framework for organization-aware agents”. Submitted to the journal of Autonomous Agents and Multi-Agent Systems.

Jensen, A. S., Kaysø-Rørdam, C., and Villadsen, J. (2015b). “Interfacing agents to real-time strategy games”. Submitted to The Thirteenth Scandinavian Conference on Artificial Intelligence, SCAI 2015.

Jensen, A. S., Spurkeland, J. S., and Villadsen, J. (2013b). “Formalizing theatrical performances using multi-agent organizations”. In: *Twelfth Scandinavian Conference on Artificial Intelligence, SCAI 2013, Aalborg, Denmark, November 20-22, 2013*. Ed. by Jaeger, M. et al. Vol. 257. IOS Press, pp. 135–144.

Jones, A. J. and Sergot, M. (1993). “On the characterisation of law and computer systems: the normative systems perspective”. In: *Deontic Logic in Computer Science: Normative System Specification*. Ed. by Meyer, J.-J. and Wieringa, R. J. John Wiley & Sons, pp. 275–307.

Jones, A. J. and Sergot, M. (1996). “A formal characterisation of institutionalised power”. In: *Logic Journal of IGPL* 4.3, pp. 427–443.

Jongmans, S.-S. T. Q., Hindriks, K. V., and Riemsdijk, M. B. van (2010). “Model checking agent programs by using the program interpreter”. In: *Computational Logic in Multi-Agent Systems*. Ed. by Dix, J. et al. Vol. 6245. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 219–237.

Juan, T., Pearce, A., and Sterling, L. (2002). “ROADMAP: extending the gaia methodology for complex open systems”. In: *The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy, Proceedings*. ACM, pp. 3–10.

Kaysø-Rørdam, C. (2014). “Implementing Intelligent Agents in Games”. MSc thesis. Technical University of Denmark.

Klapiscak, T. and Bordini, R. H. (2009). “Jasdl: a practical programming approach combining agent and semantic web technologies”. In: *Declarative Agent Languages and Technologies VI*. Ed. by Baldoni, M. et al. Vol. 5397. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 91–110.

Kumar, S., Huber, M. J., Cohen, P. R., and Mcgee, D. R. (2002). “Toward a formalism for conversation protocols using joint intention theory”. In: *Computational Intelligence* 18.2, pp. 174–228.

Li, X., Mao, W., Zeng, D., and Wang, F.-Y. (2008). “Agent-based social simulation and modeling in social computing”. English. In: *Intelligence and Security Informatics*. Ed. by Yang, C. C. et al. Vol. 5075. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 401–412.

Ljunberg, M. and Lucas, A. (1992). “The OASIS air traffic management system”. In: *Proceedings of the 2nd Pacific Rim International Conference on AI (PRICAI’92)*.

Meneguzzi, F. R. and Luck, M. (2009). “Norm-based behaviour modification in bdi agents”. In: *8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), Budapest, Hungary, May 10-15, 2009, Volume 1*. Ed. by Sierra, C. et al. IFAAMAS, pp. 177–184.

Moreno, A. and Garbay, C. (2003). “Software agents in health care”. In: *Artificial Intelligence in Medicin* 103.3, pp. 229–232.

Muscettola, N., Nayak, P. P., Pell, B., and Williams, B. C. (1998). “Remote agent: to boldly go where no ai system has gone before”. In: *Artificial Intelligence* 103.1-2, pp. 5–47.

Noriega, P. (1997). “Agent mediated auctions: the fishmarket metaphor”. PhD thesis. Universitat Autònoma de Barcelona.

Omicini, A., Ricci, A., and Viroli, M. (2008). “Artifacts in the A&A meta-model for multi-agent systems”. In: *Autonomous Agents and Multi-Agent Systems* 17.3, pp. 432–456.

Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., and Tummolini, L. (2004). “Coordination artifacts: environment-based coordination for intelligent agents”. In: *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 19-23 August 2004, New York, NY, USA*. IEEE Computer Society, pp. 286–293.

Oxford Dictionaries (Accessed online, Januar 9, 2015). *Norm*. Oxford University Press.

Padgham, L. and Lambrix, P. (2005). “Formalisations of capabilities for BDI-agents”. In: *Autonomous Agents and Multi-Agent Systems* 10.3, pp. 249–271.

Pavón, J., Gómez-Sanz, J. J., and Fuentes, R. (2005). “The INGENIAS methodology and tools”. In: *Agent-oriented methodologies* 9, pp. 236–276.

Plotkin, G. D. (1981). *A structural approach to operational semantics*. Tech. rep. University of Aarhus.

Pokahr, A., Braubach, L., and Lamersdorf, W. (2005). “Jadex: a BDI reasoning engine”. In: *Multi-Agent Programming*. Ed. by Bordini, R. et al. Vol. 15. Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer US, pp. 149–174.

Ranathunga, S., Cranefield, S., and Purvis, M. (2012). “Integrating expectation monitoring into BDI agents”. English. In: *Programming Multi-Agent Systems*. Ed. by Dennis, L. et al. Vol. 7217. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 74–91.

Rao, A. S. (1996). “AgentSpeak (L): BDI agents speak out in a logical computable language”. In: *Agents Breaking Away*. Ed. by Velde, W. van de and Perram, J. W. Vol. 1038. Lecture Notes in Computer Science. Springer Berlin Heidelberg.

Rao, A. S. and Georgeff, M. P. (1991). “Modeling rational agents within a bdi-architecture”. In: *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR’91). Cambridge, MA, USA, April 22-25, 1991*. Ed. by Allen, J. F. et al. Morgan Kaufmann, pp. 473–484.

Rao, A. S. and Georgeff, M. P. (1995). “BDI agents: from theory to practice”. In: *Proceedings of the First International Conference on Multiagent Systems (ICMAS’95), June 12-14, 1995, San Francisco, California, USA*. Ed. by Lesser, V. R. and Gasser, L. The MIT Press, pp. 312–319.

Rao, A. S. and Georgeff, M. P. (1998). “Decision procedures for BDI logics”. In: *Journal of Logic and Computation* 8.3, pp. 293–342.

Ricci, A., Viroli, M., and Omicini, A. (2006). “Programming MAS with artifacts”. In: *Programming Multi-Agent Systems*. Ed. by Bordini, R. et al. Vol. 3862. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 206–221.

Ricci, A., Viroli, M., and Omicini, A. (2007). “Cartago: a framework for prototyping artifact-based environments in mas”. In: *Environments for Multi-Agent Systems III*. Ed. by Weyns, D. et al. Vol. 4389. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 67–86.

Riemsdijk, M. B. van, Dignum, V., Jonker, C. M., and Aldewereld, H. (2011). “Programming role enactment through reflection”. In: *Proceedings of the 2011 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2011, Campus Scientifique de la Doua, Lyon, France, August 22-27, 2011*. Ed. by Boissier, O. et al. IEEE Computer Society, pp. 133–140.

Riemsdijk, M. B. van, Hindriks, K. V., and Jonker, C. (2009). “Programming organization-aware agents”. In: *Engineering Societies in the Agents World X*. Ed. by Aldewereld, H. et al. Vol. 5881. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 98–112.

Russell, S. J. and Norvig, P. (2010). *Artificial Intelligence - A Modern Approach*. 3rd international edition. Pearson Education.

Searle, J. (1995). *The Construction of Social Reality*. Free Press.

Shoham, Y. (1993). “Agent-oriented programming”. In: *Artificial Intelligence* 60.1, pp. 51–92.

Sistla, A. P., Vardi, M. Y., and Wolper, P. (1987). “The complementation problem for büchi automata with applications to temporal logic”. In: *Theoretical Computer Science* 49.2, pp. 217–237.

Spurkeland, J. S. (2013). “Interaction in Organization-Oriented Multi-Agent Systems”. MSc thesis. Technical University of Denmark.

Thangarajah, J., Padgham, L., and Winikoff, M. (2003). “Detecting & avoiding interference between goals in intelligent agents”. In: *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI’03)*. Ed. by Gottlob, G. and Walsh, T., pp. 721–726.

The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy, Proceedings (2002). ACM.

Tinnemeier, N. (2011). “Organizing agent organizations: syntax and operational semantics of an organization-oriented programming language”. PhD thesis. Utrecht University.

Torre, L. van der and Tan, Y.-H. (1999). “Contrary-to-duty reasoning with preference-based dyadic obligations”. In: *Annals of Mathematics and Artificial Intelligence* 27.1-4, pp. 49–78.

Viganò, F. (2007). “A framework for model checking institutions”. In: *Model Checking and Artificial Intelligence*. Ed. by Edelkamp, S. and Lomuscio, A. Vol. 4428. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 129–145.

Visser, W., Havelund, K., Brat, G., Park, S., and Lerda, F. (2003). “Model checking programs”. In: *Automated Software Engineering* 10.2, pp. 203–232.

Westra, J. (2011). “Organizing adaptation using agents in serious games”. PhD thesis. Utrecht University.

Winikoff, M. (2007). “Implementing commitment-based interaction”. In: *6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007), Honolulu, Hawaii, USA, May 14-18, 2007*. Ed. by Durfee, E. H. et al. IFAAMAS, pp. 868–875.

Wooldridge, M. J. (2009). *An Introduction to MultiAgent Systems*. 2nd edition. Wiley.

Wright, G. H. von (1951). “Deontic logic”. In: *Mind* 60, pp. 1–15.

Zambonelli, F., Jennings, N. R., and Wooldridge, M. (2005). “Multi-agent systems as computational organizations: the Gaia methodology”. In: *Agent-oriented Methodologies* 6, pp. 136–171.

{2.7182818284}ϑφέρτυθιοπσδφγηξκλ